intel

APPLICATION NOTE

AP-180

May 1984

Graphics Applications of the iSBX™ 275 Graphics Controller

Brad Janeway

Order Number: 280008-001

GRAPHICS APPLICATIONS OF THE iSBX™ 275 GRAPHICS CONTROLLER

CONTENTS

GENERAL INTRODUCTION

The iSBX™ 275 video graphics controller is a double-wide iSBX MULTIMODULE™ board. (Refer to Fig. 1.) It provides low-cost graphics capability for boards and systems which support the iSBX bus interface. This graphics controller simplifies graphics drawing for the user while offloading the central processing unit (CPU) of the system. Applications for the iSBX 275 graphics controller include video displays for industrial operator stations, engineering work stations, videotex, business presentation systems, and other information display systems.

The purpose of this application note is to show how easily graphics drawing can be implemented using the iSBX 275 graphics controller. For this discussion, familiarity with Intel products is assumed, including iRMX™ 86 file structure, PL/M-86 code generation, and Intel boards and board jumpering procedure. A general knowledge of algebra and geometry is also assumed, but prior graphics experience is not required.

A dual function onboard processor controls the iSBX 275 board. (Refer to Fig. 1.) This processor, the 82720 Graphics Display Controller (GDC), is both a graphics drawing processor and a display processor. The iSBX 275 board provides 16K words (32K bytes) of onboard static RAM display memory for use by the GDC. All calculations and addressing for drawing the requested figure in display memory are performed by the GDC. The GDC also maps this display memory onto the monitor screen and performs screen refresh. Refer to the 82720 GRAPHICS DISPLAY CONTROLLER Data Sheet, Intel order number 210655-002, for information on the GDC component.

The iSBX 275 board provides the capability for drawing points, lines, arcs, circles, and rectangles in a user-defined line style. It also provides for drawing characters and special user-defined symbols. In addition, rectangular areas can be filled with a user-specified pattern.

Jumpers on the iSBX 275 board enable operation in either black-and-white or eight-color display mode. The iSBX 275 board supports a black-and-white display resolution of 512 × 512 or a color display resolution of 256 × 256. For jumper configuration information, refer to the iSBX 275 VIDEO GRAPHICS CONTROLLER MULTIMODULE BOARD REFERENCE MANUAL, Intel order number 144829-001, pages 2-4 through 2-7.

GENERAL OVERVIEW

This application note begins with some background information related to raster-scan graphics. It then describes a set of PL/M-86 procedures which provide a simple interface to the iSBX 275 board. Next, two graphics applications are discussed which utilize these graphics procedures. The first application uses the black-and-white display mode of the iSBX 275 board, while the second utilizes the color display mode of the board. Finally, some performance considerations are addressed.

Listings of the applications described in this application note are contained in an appendix. The complete source code for the graphics procedures and the airplane application are available from INSITE.

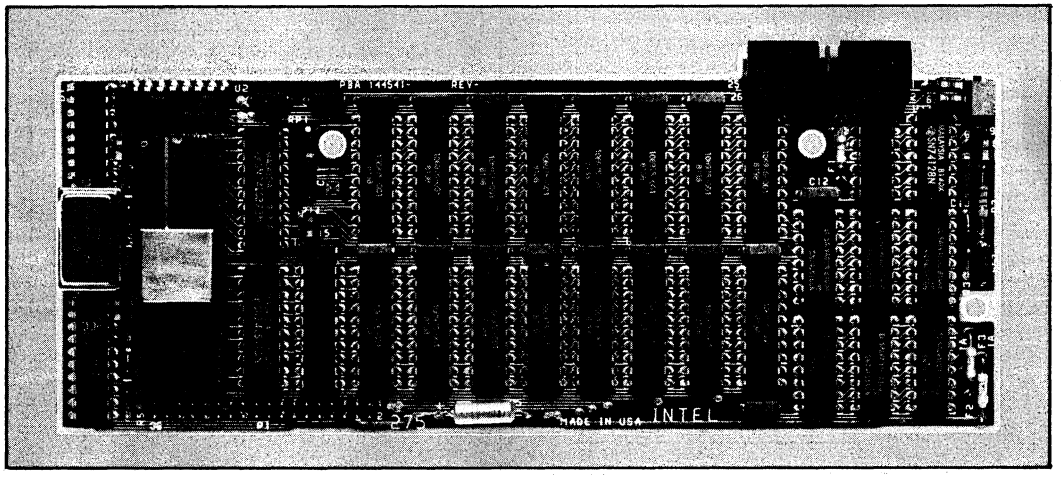


Figure 1. iSBX 275 Graphics Controller

BACKGROUND INFORMATION

Overview

This section first provides information on bit-mapped graphics. Then raster-scan monitors are discussed.

Bit-Mapped Graphics

A monitor screen can be thought of as a rectangular grid of picture elements, or pixels. This grid is defined by an aspect ratio x:y, where x is the horizontal dimension and y is the vertical dimension, in pixels. The typical monitor aspect ratio is 4:3.

An RGB (Red-Green-Blue) color monitor has three electron guns, one for each of the primary colors red, green, and blue. These guns are used for drawing on the monitor screen. A black-and-white monitor has one gun.

The iSBX 275 board contains 16K words or 256K bits of bit-mapped display memory. The GDC addresses this memory, often referred to as the bit map, as 16K words of memory. (See Fig. 2.) Each bit (or combination of 3 bits in color display mode) is mapped by the GDC to a unique pixel on a monitor screen. In black-and-white display mode, all 256K bits lie in one plane, so each bit represents one pixel. In color display mode, the bit map is divided into four color planes, with 4K words (64K bits) in each plane. These planes logically overlap each other, as shown in Fig. 3. One of these planes is used for each of the primary colors red, blue, and green. The bits from the red color plane are input to the red electron gun of an RGB monitor. The bits from the other two color planes are likewise input to the respective guns of the monitor. The color of a pixel on a monitor screen is thus determined by three bits, one from each plane. This results in a palette of eight drawing colors, ranging from all three bits of a pixel OFF (black) to all three bits ON (white).

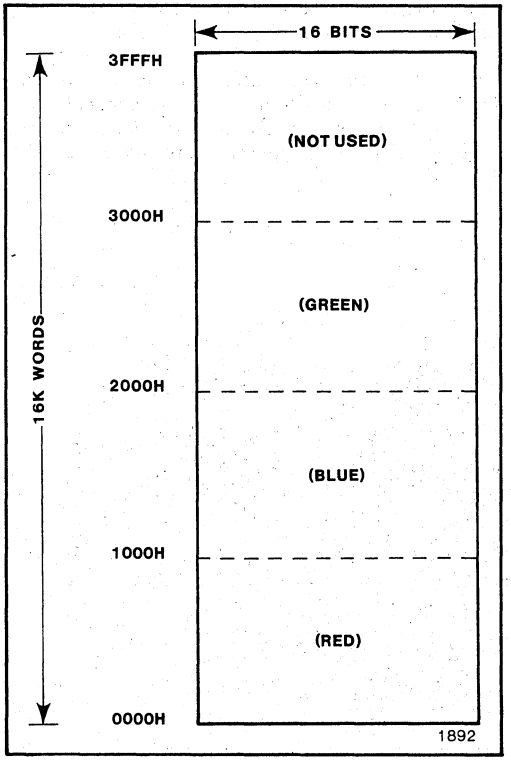


Figure 2. GDC Addressing

In order for the GDC to properly map bits in display memory onto the monitor screen, the bit map must be configured to match the desired display image. Configuring the bit map for a 1:1 aspect ratio provides the most efficient use of the display memory on the iSBX 275 board, as all of the bits in the display memory are utilized. As shown in Fig. 4a and Fig. 5a, this results in a black-and-white resolution of 512 × 512 or a color resolution of 256 × 256 pixels. However, an aspect ratio of 4:3 is typically desired, and any reasonable aspect ratio may be used. In any configuration, the horizontal dimension must be a multiple of 32 pixels. The black-and-white configuration shown in Fig. 4b leaves a minimum of 64 bits (4 words) in the bit map unused, and the color configuration shown in Fig. 5b leaves a minimum of 160 bits (10 words) unused. In these figures, the shaded region represents an exact 4:3 aspect ratio, resulting in

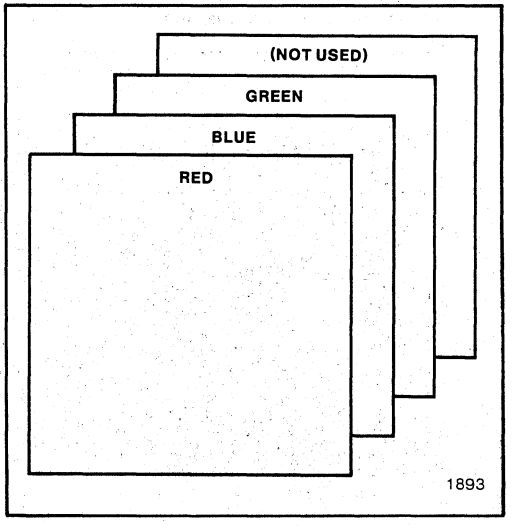


Figure 3. Color-Plane Orientation

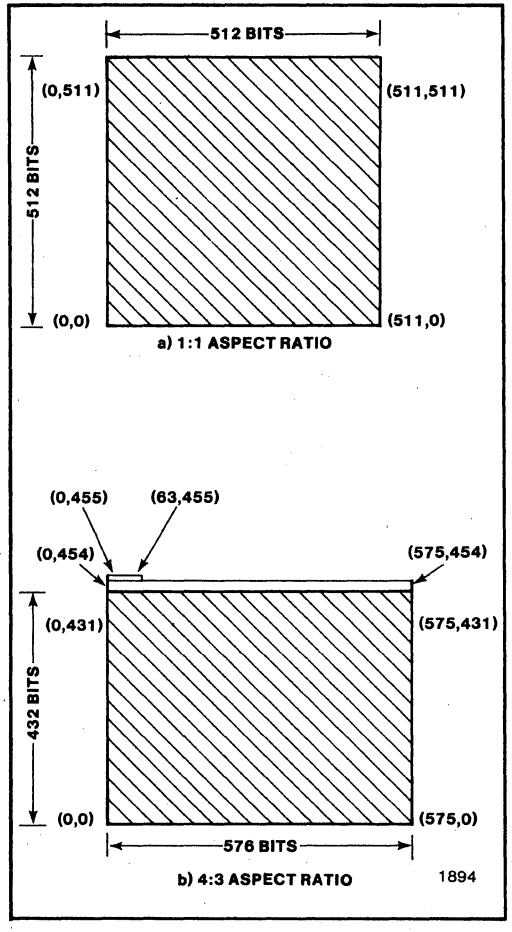


Figure 4. Black-and-white Mode Configuration

13312 bits (832 words) or 3328 bits (208 words) remaining unused for the black-and-white or color configuration, respectively.

A logical x,y address is associated with each drawing command sent by user code to the iSBX 275 board. Fig. 4 and Fig. 5 show the range of logical addresses for some typical bit map configurations. The logical coordinates associated with a drawing command are translated into the appropriate bit map word address (or addresses, in color mode). This is accomplished by the iSBX 275 routines described in the PL/M-86 GRAPHICS LIBRARY section. The iSBX 275 board then sets the appropriate display memory bits.

Raster-Scan Monitors

The iSBX 275 board utilitzes and RGB monitor for color displays and either an RGB or a black-and-white monitor for monochrome displays. An RGB monitor has three phosphors and three electron guns, one pair for each of the primary colors red, blue, and green, for generating images on its screen. For a color display using the iSBX 275 board, the bits from the red, blue, and green color planes in display memory are fed to the respective electron gun inputs of an RGB monitor. For a black-and-white display using and RGB monitor, the bits from display memory are typically fed to the green gun input of the monitor. A black-and-white monitor has one electron gun and one phosphor for generating images on its screen.

A black-and-white monitor typically has two synchronization inputs, one for horizontal sync and one for vertical sync. An RGB monitor typically has one sync input, called Combined SYNC (CSYNC), which is a combination of these two signals.

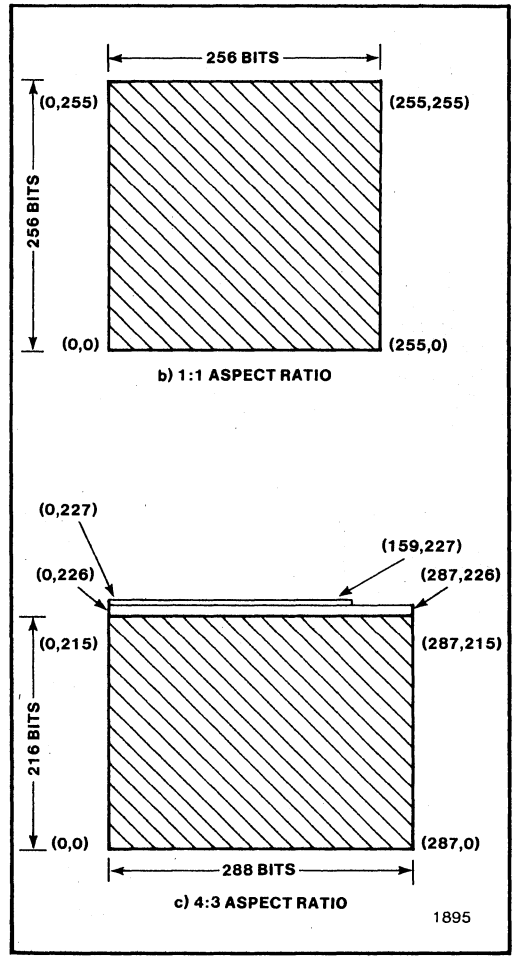


Figure 5. Color Mode Configuration

The following discussion deals with the manner in which a raster-scan monitor generates an image on its screen. Monitor timing is described, providing an understanding of the initialization parameters which are required by the iSBX 275 board. For simplicity, the discussion assumes a black-and-white monitor.

CRT

The inside surface of a cathode ray tube (CRT) is coated with a light-emitting phosphor. A raster-scan monitor displays an image by scanning an electron gun across the face of the CRT, moving from left to right and from top to bottom (as viewed from the front of the monitor). For each pixel which is to be turned ON, the gun generates an electron beam of sufficient energy to cause the pixel to glow brightly. However, the persistence of the phosphor is typically only twenty milliseconds, requiring that the phosphor be struck at regular intervals in order to keep it glowing brightly. This is accomplished by continuously scanning the entire face of the CRT as described below.

RASTER-SCAN

A complete display image on a monitor screen is called a frame. It consists of one or two fields of information, each generated by one raster-scan cycle. (Refer to the section on INTERLACING.) A raster-scan cycle consists of a vertical retrace-and-scan and a number of horizontal retrace-and-scans. Upon receiving a vertical syncronization pulse from the iSBX 275 board, a monitor begins a raster-scan cycle. The beginning of a frame is defined by the coincidence of a vertical and a horizontal sync pulse from the iSBX 275 board.

In response to a vertical sync pulse, a monitor first executes a vertical retrace, moving its electron gun rapidly to the top of the CRT. Then the monitor performs a vertical scan. It moves its electron gun back down the face of the CRT at a constant, slower rate, arriving at the bottom just as the next vertical sync pulse arrives. The monitor expects a vertical sync pulse at regular intervals, defined by the field rate of the monitor. The typical field rate is 60 Hz in the U.S. and 50 Hz in Europe. The length of the sync pulse must be within the limits specified by the monitor.

Upon receiving a horizontal sync pulse, a monitor first executes a horizontal retrace, moving its electron gun rapidly to the left of the CRT. The monitor then performs a horizontal scan. It moves its electron gun back across the face of the CRT at a constant, slower rate, arriving at the right side just as the next horizontal sync pulse arrives. The monitor expects a horizontal sync pulse at regular intervals, defined by the horizontal rate of the monitor. The typical horizontal rate is 15.75 KHz in the U.S. and 15.63 KHz in Europe. The length of the sync pulse must be within the limits specified by the monitor.

The path which the electron gun traces across the face of the CRT, due to the combination of vertical and horizontal retrace-and-scan cycles, is similar to that shown in Fig. 6 and Fig. 7. The dotted lines represent retrace and the solid lines represent horizontal scan. All of the horizontal scan lines traced during a vertical scan comprise a field. The number of lines in a field is a function of the field rate and the horizontal rate of the monitor. Fig. 6 shows the case for which a frame consists of one field, and Fig. 7 shows the case for which a frame consists of two interlaced fields.

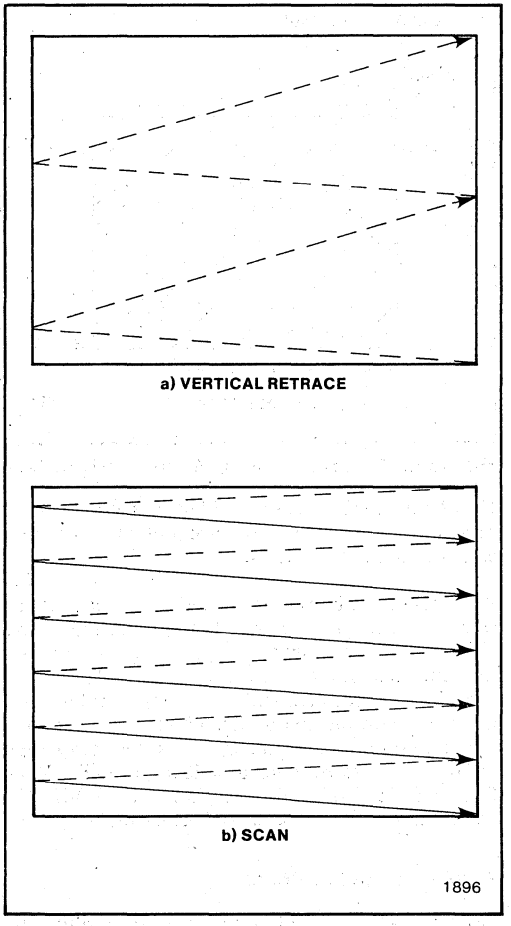


Figure 6. Non-interlaced Raster Scan Frame = Field

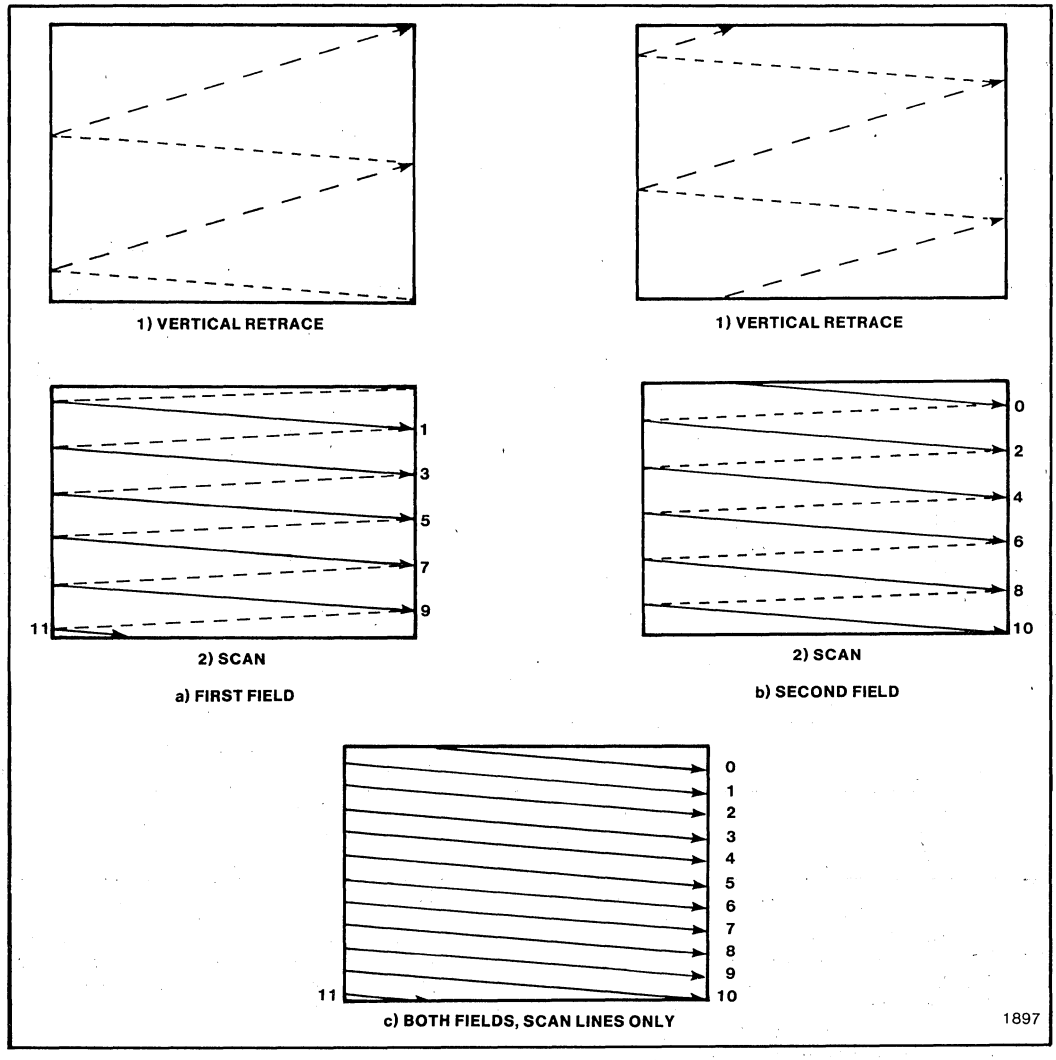


Figure 7. Interlaced Raster Scan Frame = 2 Fields

During the active diplay portion of each horizontal scan, the display memory bits mapped to that scan line are input to the electron gun of the monitor. The electron gun interprets each bit as beam intensity. A logical "1" bit causes the gun to excite the phosphor at that pixel location, such that it emits light, whereas a logical "0" bit leaves the pixel OFF. The rate at which these bits are input to the gun, and hence the rate at which pixels are drawn on the monitor screen, is referred to as the bandwidth of the board. For the iSBX 275 board, this bandwidth is determined by the frequency of the oscillator on the board. In the divide-by-two mode of the iSBX 275 board, this bandwidth is half of the oscillator frequency. The board bandwidth must be within the bandwidth range of the monitor.

In order to avoid leaving unwanted traces across the display, the electron gun must be off, or blanked, during retrace. Since a discrete time period is required to turn off an electron beam, a monitor specifies minimum horizontal and vertical front porch blanking times. (Refer to Fig. 8.) During a front porch blanking time, which immediately precedes the corresponding sync pulse, the monitor turns off its electron gun. The front porch

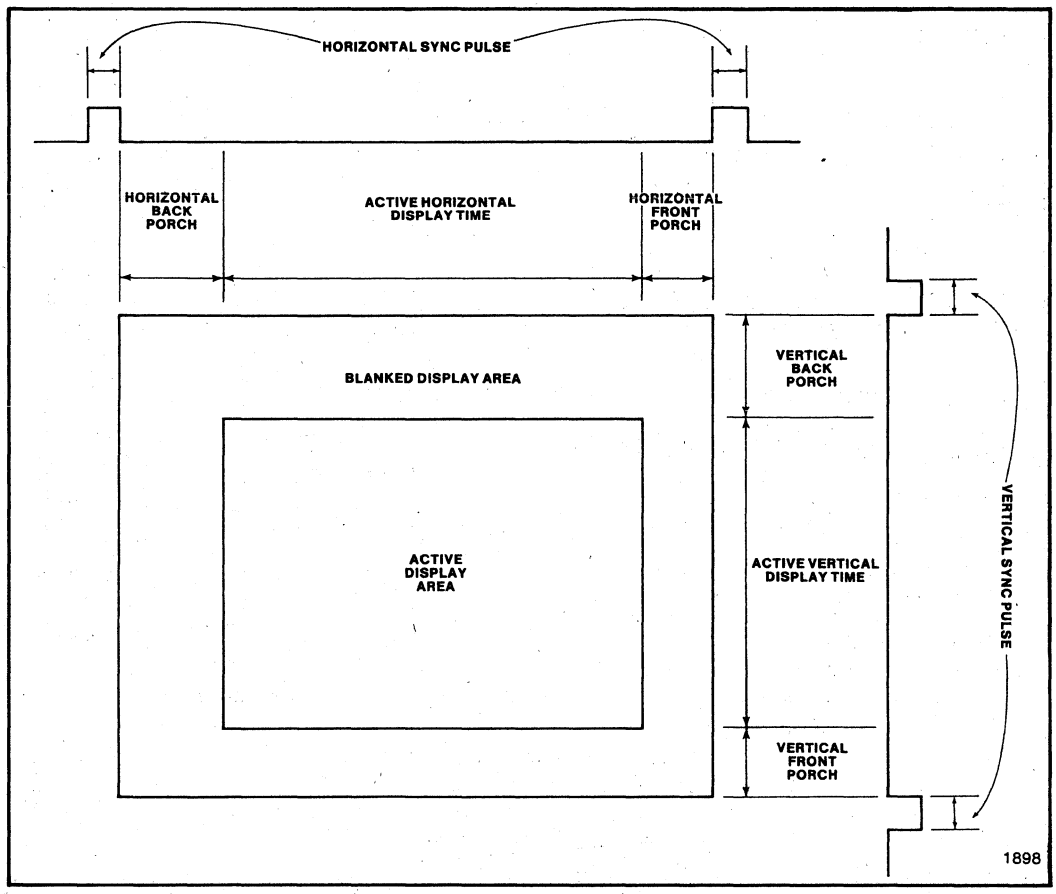


Figure 8. Monitor Timing

guarantees that the electron beam will be blanked by the time the sync pulse arrives. During the sync pulse blanking time, the electron gun retraces back to the left and/or top of the CRT. Following retrace, a discrete time period is required to position the electron gun at the first pixel of the new raster line. Therefore, a monitor also specifies minimum horizontal and vertical back porch blanking times, which immediately follow the corresponding sync pulse. Thus, horizontal or vertical blanking time is defined by the respective front porch, sync pulse, and back proch. The iSBX 275 board does not input any display memory bits to the electron gun during blanking time.

Fig. 8 shows the relationship of active display time and blanking time to the entire raster-scan area. Increasing the horizontal front porch, sync, or back porch decreases the number of pixels which are displayed in the horizontal direction. Increasing the vertical values decreases the number of lines which are displayed. These values should therefore be set such that the active display area exactly matches the desired bit map configuration. The active display area then contains only those pixels which are mapped from the display memory on the iSBX 275 board.

INTERLACING

The horizontal resolution of a monitor, or the number of pixels in each display line, is a function of the bandwidth and the horizontal rate of the monitor, as described above. Given the monitor horizontal rate, a higher bandwidth means more pixels per line. The iSBX 275 board, with the 12.6 MHz default oscillator, provides a choice of two bandwidths, one twice the rate of the other. The slower rate is used for non-interlaced display mode operation. When interlaced display mode operation is selected, doubling the number of lines per inch, the bandwidth must also be doubled.

The vertical resolution of a monitor, or the number of lines displayed, is a function of the field rate and the horizontal rate of the monitor. Given these two rates, the vertical resolution is fixed. It can be increased then only through interlacing, which doubles the vertical resolution.

In non-interlaced display mode operation, the electron gun always begins and ends a raster-scan cycle in the lower right corner of the CRT, as shown in Fig. 6. A frame is thus comprised of one field, as each field is scanned directly over the preceding one. In interlaced display mode operation, a frame is comprised of two interlaced fields, as shown in Fig. 7. The electron gun always begins the first field and ends the second field in the lower right corner of the CRT. This is accomplished by scanning an odd number of lines for each frame. Since an odd number of lines cannot be equally divided between two fields, each field receives part of the extra line. The first field (Fig. 7a) begins its vertical retrace in the lower right corner and ends its vertical scan partway through the odd horizontal scan. The second field (Fig. 7b) then starts its vertical retrace partway through the odd horizontal scan and ends its vertical scan in the lower right corner. Due to the slanted nature of the scan lines, the two fields are vertically offset from each other by half of a line. (Refer to Fig. 7c.) Interlacing doubles the vertical resolution of the monitor by scanning twice as many lines per inch, but the frame rate is half of the non-interlaced display mode frame rate.

PL/M-86 GRAPHICS LIBRARY

Introduction

The graphics procedures described below provide a simple interface to the basic capabilities of the iSBX 275 board. These procedures require an iSBC® 337 Numeric Data Processor MULTIMODULE board or equivalent in the system. For this application note, the graphics procedures were placed into a library with the pathname /user/sbx275/lib/sbx275. Each module calling any of these procedures contained the following include file:

```
$include(/user/sbx275/ext/sbx275)
```

This include file contains all of the external declarations of the graphics procedures, as well as the literal declarations for all of the "parameter choices" available to the procedures. These "parameter choices" are listed below, as well as in the appropriate procedure descriptions.

```
zeros, ones
bw, red, blue, magenta, green, yellow, cyan,
  white
south, southeast, east, northeast, north,
  northwest, west, southwest
bw_plane, red_plane, blue_plane, green_plane
normal_char, slant_char
replace, complement, reset, set
one_x, two_x, three_x, four_x, five_x, six_x,
  seven_x, eight_x, nine_x, ten_x,
  eleven_x, twelve_x, thirteen_x, fourteen_x,
  fifteen_x, sixteen_x
```

"Parameter Choice" Literals

For addressing purposes, the monitor screen is in the first quadrant of the Cartesian plane. The origin is in the lower left corner of the display, with positive x to the right and positive y up.

Overview

First, the graphics procedures are divided into several functional groups and briefly defined, with those in each group listed alphabetically. This provides an easy reference to the procedures rather than a detailed description of them. Then all of the parameters used by these procedures are defined, in alphabetical order. The parameter definitions provide a more extensive description of the capabilities of these procedures. Except for those parameters defined as pointers or byte variables, all parameters are 16-bit word values.

Initialization Procedures

Of the following routines, only INIT must be called by the applications programmer. The rest are called as a result of the INIT call.

ABCMD: causes the GDC to exit idle mode and enable (begin) the display.

ABCMD must be called following an ARCMD call, in order to unblank the display.

Calls: CMDOUT.

Called by UICMD.

ACCMD: disables the cursor.

Calls: CMDOUT, DATOUT.

Called by UICMD.

ARCMD(reset_parameters_pointer): resets the GDC and puts it in idle mode, blanking the display.

Calls: CMDOUT, DATOUT.

Called by UICMD.

AVCMD(1): sets the GDC as the master vertical sync generator.

Calls: CMDOUT.

Called by UICMD.

INIT(plane_size,xres,yres,hs,vs,hfp,hbp,vfp, vbp,opmode,command_port,data_port):

assigns the bit map, monitor, and operation mode parameters;
assigns the iSBX 275 board I/O port addresses;
calls UICMD, initializing the iSBX 275 board;
calls UPCMD and AWCMD, clearing display memory;
calls AGCMD, initializing for solid area fill and line drawing;
calls UMCMD, setting the drawing color to BW and the drawing mode to SET.

INIT must be the first iSBX 275 board procedure called by a user program. It need only be called once for a given application program.

UICMD: calls INITFP from 8087.LIB, initializing the 8087;
calls ARCMD, resetting the GDC and putting it in idle mode;
calls APCMD, setting the pitch equal to the display area width;
calls AZCMD, setting the zoom magnification factor to ONE_X;
calls ACCMD, disabling the cursor display;
calls ASCMD, setting up a single display window which starts at address 0 and comprises the entire display area;
calls AVCMD, setting the GDC as the master video sync generator;
calls ABCMD, beginning the display.

Called by INIT.

Control Procedures

These routines provide control for drawing in the bit map.

AGCMD(pattern7_8,pattern5_6,pattern3_4, pattern1_2): loads an 8 × 8 bit drawing pattern.

INIT loads a pattern for solid area fill, which also defines solid line drawing.

Calls: CMDOUT, DATOUT.

Called by INIT.

APCMD(words): sets the pitch (bit map width).

INIT sets the pitch equal to the display area width, which is (xres÷16) display memory words.

In black-and-white, non-interlaced display mode, the bit map typically will be larger than the display area. With the pitch equal to (xres÷16), the bit map will be longer than the display area. If an APCMD call sets the pitch greater than (xres÷16), the bit map will be at least wider and possibly also longer than the display area. Panning and scrolling of the display windows over this larger bit map are accomplished with ASCMD.

Calls: CMDOUT, DATOUT.

Called by UICMD.

ASCMD(start_address1,length1,start_address2, length2): defines the top and bottom display windows into which the display area can be divided.

INIT sets up only one display window, which starts at address 0 and comprises the entire display area.

In black-and-white, non-interlaced display mode, ASCMD can be used to pan or scroll the display windows independently over the bit map. If an APCMD call has set the pitch wider than the display area, the display windows can be panned across the bit map. Panning is accomplished by a series of ASCMD calls which increment or decrement, by one, the starting address of a window. Panning a display window beyond the left or right borders of the bit map will distort the displayed image. If the bit map is longer than the display area, the display windows can be scrolled over the bit map. Scrolling is accomplished by a series of ASCMD calls which increment or decrement, by the pitch, the starting address of a window. Scrolling a display window beyond the top or bottom borders of the bit map may not be desireable.

Calls: CMDOUT, DATOUT.

Called by UICMD.

AWCMD (write_mode , direction , aw_pattern, aw_count): writes multiple words of all zeros or all ones directly to display memory.

An AWCMD call must be preceded by a UPCMD call. The following sequence will clear display memory:

```
call UPCMD(0,0,BW_PLANE);
call AWCMD ( REPLACE , EAST ,
                    ZEROS, 4000H);
```

Calls: CMDOUT, DATOUT.

Called by INIT.

AZCMD(zoom_factor): sets the drawing magnification factor.

INIT sets zoom_factor to ONE_X.

Calls: CMDOUT, DATOUT.

Called by UICMD, STRCMD.

UMCMD(color,write_mode): sets the drawing color and mode.

INIT sets color to BW and write_mode to SET.

Called by INIT, STRCMD.

UPCMD(x1,y1,plane): positions the cursor in display memory.

Calls: POSCUR.

Called by INIT.

Figure Drawing Procedures

These routines draw figures in the bit map.

UACMD(xc,yc,radius,direction,angle_T,angle_S): draws an arc.

Calls: CAP, CMDOUT, POSCUR, LPARMS, DFIG.

UBCMD(x1,y1,length,height,direction): draws a rectangle (box).

Calls: CMDOUT, POSCUR, LPARMS, DFIG.

UCCMD(xc,yc,radius): draws a circle.

Calls: CAP, CMDOUT, POSCUR, LPARMS, DFIG.

UFCMD(x1,y1,length,height,direction): fills a rectangular area with the pattern specified by the most recent AGCMD call.

Calls: CMDOUT, POSCUR, LPARMS.

ULCMD(x1,y1,x2,y2): draws a line.

Calls: CMDOUT, POSCUR, LPARMS, DFIG.

USCMD(x1,y1,direction): draws the symbol defined by the most recent AGCMD call.

Calls: CMDOUT, POSCUR, LPARMS.

Text Drawing Procedures

These routines draw ASCII characters in the bit map.

STRCMD(x1,y1,slant,direction,zoom_factor,color, text_length,text_pointer): draws an ASCII string.

An STRCMD call alters the fill and draw pattern, so AGCMD must be called to reload the pattern before resuming figure drawing.

An STRCMD call sets zoom_factor to ONE_X before returning.

Calls: AZCMD, UMCMD, UDCMD.

UDCMD(x1,y1,slant,direction,character): draws an ASCII character.

A UDCMD call alters the fill and draw pattern, so AGCMD must be called to reload the pattern before resuming figure drawing.

Calls: CMDOUT, DATOUT, POSCUR, LPARMS.

Support Procedures

None of these routines would generally be called by an applications programmer. They are called by the routines described in the previous procedure groups.

CAP(radius,angle_S,direction,angle_T,arc_parameter_pointer): calculates arc drawing parameters.

Calls: mqerSIN, mqerIE2 from CEL87.LIB.

Called by UACMD, UCCMD.

CMDOUT(command_byte): writes a command to the GDC command port and logs it in mem$buf, a 256-byte circular buffer.

Called by ABCMD, ACCMD, AGCMD, APCMD, ARCMD, ASCMD, AVCMD, AWCMD, AZCMD, UACMD, UBCMD, UCCMD, UDCMD, UFCMD, ULCMD, USCMD, DFIG, LPARMS, POSMP.

DATOUT(data_byte): writes a data byte to the GDC data port and logs it in mem$buf, a 256-byte circular buffer.

Called by ACCMD, AGCMD, APCMD, ARCMD, ASCMD, AWCMD, AZCMD, UDCMD, LPARMS, POSMP.

DFIG: starts the GDC drawing a figure.

Calls: CMDOUT.

Called by UACMD, UBCMD, UCCMD, ULCMD.

LPARMS(type_and_direction,dc,d,d2,d1,dm): loads the figure drawing parameters to the GDC.

Calls: CMDOUT, DATOUT.

Called by UACMD, UBCMD, UCCMD, UDCMD, UFCMD, ULCMD, USCMD.

POSCUR(x1,y1,plane): converts a cursor address given relative to the x-y coordinate system into an absolute display memory address, which it sends to the GDC via POSMP.

Calls: POSMP.

Called by UACMD, UBCMD, UCCMD, UDCMD, UFCMD, ULCMD, UPCMD, USCMD.

POSMP(plane,word_address,dot_address): loads an absolute cursor address to the GDC.

Calls: CMDOUT, DATOUT.

Called by POSCUR.

Character Generator

CG5X7: character font table for drawing 5 × 7 characters in an 8 × 8 matrix.

Used by UDCMD, STRCMD.

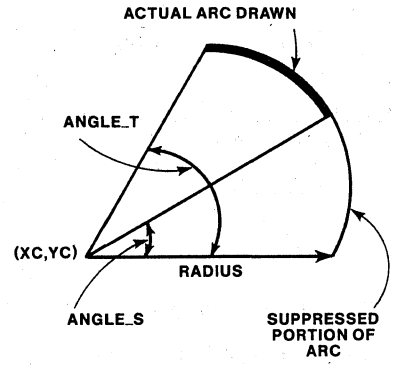
Parameter Definitions

The parameters defined here are used by the routines described briefly above.

angle_S: the angle from the major axis to the start of the arc, $0 \leq$ angle_S < 45, in degrees. (See the figure below.)

angle_T: the angle from the major axis to the end of the arc, $0 <$ angle_T ≤ 45, in degrees. (See the figure below.)

The major axis is the x or y axis with respect to which an arc angle is measured.



arc_parameters_pointer: a pointer to the table containing arc drawing parameters.

aw_count: the number of consecutive words AWCMD is to write to display memory.

aw_pattern: the 16-bit pattern AWCMD uses to write to display memory.

Parameter choices are ZEROS or ONES.

character: the ASCII code for the character to be drawn by USCMD.

color: the drawing color.

Parameter choices are BW, RED, BLUE, MAGENTA, GREEN, YELLOW, CYAN, or WHITE.

BW is white in black-and-white display mode.

Color Plane			Displayed Color
Green	Blue	Red	
0	0	0	Black
0	0	1	Red
0	1	0	Blue
0	1	1	Magenta
1	0	0	Green
1	0	1	Yellow
1	1	0	Cyan
1	1	1	White

command_byte: a byte which the GDC interprets as a command.

command_port: the address of the I/O port on the SBX baseboard through which commands will be sent to the iSBX 275 board.

The I/O port must support 8-bit MULTIMODULE™ boards. When addressed, the port must activate MCS0/ on the SBX bus.

data_byte: a parameter byte which the GDC interprets according to the relative order in which the GDC receives it, following the most recent command byte.

data_port: the address of the I/O port on the SBX baseboard through which data will be sent to the iSBX 275 board.

The I/O port must support 8-bit MULTIMODULE boards. When addressed, the port must activate MCS0/ on the SBX bus.

dc,d,d2,d1,dm: five figure drawing parameters which are interpreted by the GDC according to the drawing type, as described below.

DRAWING TYPE	DC	D	D2	D1	DM
INITIAL VALUE*	0	8	8	−1	−1
LINE	$\lvert\triangle\rvert$	$2\lvert\triangle D\rvert-\lvert\triangle\rvert$	$2(\lvert\triangle D\rvert-\lvert\triangle\rvert)$	$2\lvert\triangle D\rvert$	-
ARC**	$r\sin\phi\rfloor$	4−1	$2(r-1)$	−1	$r\sin\theta\rfloor$
RECTANGLE	3	A−1	B−1	−1	A−1
AREA FILL	B−1	A	A	-	-
GRAPHIC CHARACTER***	B−1	A	A	-	-
WRITE DATA	W−1	-	-	-	-
READ DATA	W	-	-	-	-

*INITIAL VALUES FOR THE VARIOUS PARAMETERS ARE LOADED WHEN THE FIGS COMMAND BYTE IS PROCESSED.
**CIRCLES ARE DRAWN WITH 8 ARCS, EACH OF WHICH SPAN 45°, SO THAT $\sin\phi = 1/\sqrt{2}$ AND $\sin\theta = 0$.
***GRAPHIC CHARACTERS ARE A SPECIAL CASE OF BIT-MAP AREA FILLING IN WHICH B AND A ≤ 8. IF A = 8 THERE IS NO NEED TO LOAD D AND D2.

WHERE:
−1 = ALL ONES VALUE.
ALL NUMBERS ARE SHOWN IN BASE 10 FOR CONVENIENCE. THE GDC ACCEPTS BASE 2 NUMBERS (2s COMPLEMENT NOTATION WHERE APPROPRIATE).
- = NO PARAMETER BYTES SENT TO GDC FOR THIS PARAMETER.
$\lvert\triangle\rvert$ = THE LARGER OF $\triangle x$ OR $\triangle y$.
$\triangle D$ = THE SMALLER OF $\triangle x$ OR $\triangle y$.
r = RADIUS OF CURVATURE, IN PIXELS.
ϕ = ANGLE FROM MAJOR AXIS TO END OF THE ARC. $\phi \leq 45°$.
θ = ANGLE FROM MAJOR AXIS TO START OF THE ARC. $\theta \leq 45°$.
$\lceil\ \rceil$ = ROUND UP TO THE NEXT HIGHER INTEGER.
$\lfloor\ \rfloor$ = ROUND DOWN TO THE NEXT LOWER INTEGER.
A = NUMBER OF PIXELS IN THE INITIALLY SPECIFIED DIRECTION.
B = NUMBER OF PIXELS IN THE DIRECTION AT RIGHT ANGLES TO THE INITIALLY SPECIFIED DIRECTION.
W = NUMBER OF WORDS TO BE ACCESSED.
DC = DRAWING COUNT PARAMETER WHICH IS ONE LESS THAN THE NUMBER OF RMW CYCLES TO BE EXECUTED.
DM = DOTS MASKED FROM DRAWING DURING ARC DRAWING.

direction: the initial drawing direction.

Parameter choices are SOUTH, SOUTHEAST, EAST, NORTHEAST, NORTH, NORTHWEST, WEST, or SOUTHWEST.

	DIR	LINE	ARC	CHARACTER	SLANT CHAR	RECTANGLE
SOUTH:	000					
SOUTHEAST:	001					
EAST:	010					
NORTHEAST:	011					
NORTH:	100					
NORTHWEST:	101					
WEST:	110					
SOUTHWEST:	111					

dot_address: the portion of the absolute cursor address which specifies the bit location within the display memory word specified by word_address.

hbp: the horizontal back porch, expressed in display memory words.

The hbp parameter is dependent on hs, hfp, and AW. (See the xres definition.)

Interlaced Display Mode

For interlaced display mode, both of the following equations must be satisfied, with hbp rounded up to the nearest integer:

hbp ≥ 5, a GDC requirement; and

hbp ≥ (monitor min. hbp) × (iSBX 275 board bandwidth) ÷ 16.

Non-interlaced Display Mode

For non-interlaced display mode, all three of the following equations must be satisfied, with hbp rounded up to the nearest integer:

hbp ≥ 5, a GDC requirement if two display windows are used:

hbp ≥ 3, a GDC requirement; and

hbp ≥ (monitor min. hbp) × (iSBX 275 board bandwidth) ÷ 16.

height: the number of pixels in the dimension perpendicular to the specified drawing direction.

hfp: the horizontal front porch, expressed in display memory words.

The hfp parameter is dependent on hs, hbp, and AW. (See the xres definition.)

Interlaced Display Mode

For interlaced display mode, the following three equations must be satisfied, with hfp rounded up to the nearest integer:

hfp ≥ 6, a GDC requirement if the light pen input is used;

hfp ≥ 2;

hfp ≥ (monitor min. hfp) × (iSBX 275 board bandwidth) ÷ 16.

Non-interlaced Display Mode

For non-interlaced display mode, the following three equations must be satisfied, with hfp rounded up to the nearest integer:

hfp ≥ 6, a GDC requirement if the light pen input is used;

hfp ≥ 2;

hfp ≥ (monitor min. hfp) × (iSBX 275 board bandwidth) ÷ 16.

hs: the horizontal sync, expressed in display memory words.

The hs parameter is dependent on hfp, hbp, and AW. (See the xres definition.)

Interlaced Display Mode

For interlaced display mode, both of the following equations must be satisfied, with hs rounded up to the nearest integer:

hs ≥ 5, a GDC requirement; and

hs ≥ (monitor min. hs) × (iSBX 275 board bandwidth) ÷ 16.

Non-interlaced Display Mode

For non-interlaced display mode, the following equation must be satisfied, with hs rounded up to the nearest integer:

hs ≥ (monitor min. hs) × (iSBX 275 board bandwidth) ÷ 16.

length: the number of pixels in the specified drawing direction.

length1: the number of display lines comprising the top display window.

length2: the number of display lines comprising the bottom display window.

opmode: the operation mode of the GDC.

MSB | 0 0 0 F I 0 G S | LSB

I	S	Description
0	0	Non-interlaced
0	1	Invalid
1	0	Interlaced Repeat Field for Character Displays
1	1	Interlaced

G	Description
0	Mixed Graphics and Character Mode
1	Graphics Mode

F	Description
0	Drawing During Active Display Time and Retrace Blanking
1	Drawing Only During Retrace Blanking

pattern7_8,pattern5_6,pattern3_4,pattern1_2: the four pieces of an 8 × 8 bit pattern, used for drawing a random 8 × 8 symbol or filling a rectangular area.

The pattern is applied backwards, as shown in the example below. The example assumes that write_mode is set to REPLACE.

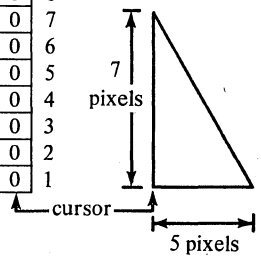
For pattern7_8 = 0602h,
pattern5_6 = 0a0eh,
pattern3_4 = 3212h,
pattern1_2 = 003eh,

the 8 × 8 bit pattern and the resulting figure displayed are as shown. The initial cursor position is designated by the arrows.

8 × 8 BIT PATTERN

0	0	0	0	0	0	1	0	8
0	0	0	0	0	1	1	0	7
0	0	0	0	1	1	1	0	6
0	0	0	0	1	0	1	0	5
0	0	0	1	0	0	1	0	4
0	0	1	1	0	0	1	0	3
0	0	1	1	1	1	1	0	2
0	0	0	0	0	0	0	0	1

FIGURE DISPLAYED

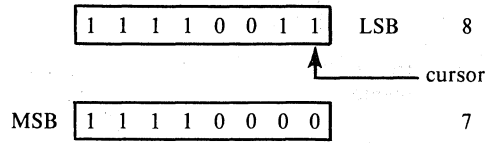


For each line, rectangle, arc, or circle drawing command issued, pattern7_8 also serves as the 16-bit drawing pattern. For each such command issued, pattern7_8 is applied repetitively from least- to most-significant bit, as shown in the example below. The example assumes that write_mode is set to REPLACE.

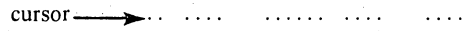
For pattern7_8 = 0f0f3h,

the 2 × 8 bit portion of the 8 × 8 bit pattern, and the resulting line pattern that is drawn, are as shown. The initial cursor position is also shown.

PATTERN7_8



LINE PATTERN



plane: the color plane in which drawing is to occur. (See the color parameter definition.)

Parameter choices are BW_PLANE, RED_PLANE, BLUE_PLANE, or GREEN_PLANE.

- BW_PLANE - starts at display memory word address 0 (for black-and-white display mode).
- RED_PLANE - starts at display memory word address 0.
- BLUE_PLANE - starts at display memory word address 1000h.
- GREEN_PLANE - starts at display memory word address 2000h.

Display memory word addresses 3000h to 3ffffh are not used in color mode.

plane_size: the number of display memory words per plane.

For black-and-white display mode, the entire display memory is used as a single-plane bit map, so

plane_size = 16,384 (or 4000h).

For color display mode, display memory is divided into four equal planes, one for each of the three primary colors and one which is unused, so

plane_size = 4096 (or 1000h).

radius: the distance, in pixels, from the center of an arc or circle to any point on that arc or circle.

reset_parameters_pointer: a pointer to the table in INIT containing the monitor and operation mode parameters for ARCMD.

slant: the character style. (See the direction parameter definition.)

Parameter choices are NORMAL_CHAR or SLANT_CHAR.

start_address1: the address of the first display memory word of the top display window.

start_address2: the address of the first display memory word of the bottom display window.

text_length: the number of ASCII bytes to be taken from the text buffer by STRCMD and drawn in the bit map.

text_pointer: a pointer to the text buffer used by STRCMD.

type_and_direction: the drawing direction and type.

MSB | S | R | A | G | L | DIR | LSB

	Drawing Type
S	Slanted Graphics Character
R	Rectangle
A	Arc/Circle
G	Graphics Character
L	Line (Vector)

A "1" enables a type.

DIR - see direction description

vbp: the vertical back porch, expressed in display lines.

The vbp parameter is dependent on vs, vfp, and AL. (See the yres definition.)

The following equation must be satisfied, with vbp rounded up to the nearest integer:

$vbp \geq$ (monitor min. vbp) × (monitor horizontal rate).

vfp: the vertical front porch, expressed in display lines.

The vfp parameter is dependent on vs, vbp, and AL. (See the yres definition.)

The following equation must be satisfied, with vfp rounded up to the nearest integer:

$vfp \geq$ (monitor min. vfp) × (monitor horizontal rate).

vs: the vertical sync, expressed in display lines.

The vs parameter is dependent on vfp, vbp, and AL. (See the yres definition.)

The following equation must be satisfied, with vs rounded up to the nearest integer:

$vs \geq$ (monitor min. vs) × (monitor horizontal rate).

words: the pitch (width of the bit map), expressed in display memory words.

word_address: the portion of the absolute cursor address which specifies the display memory word location in the color plane specified by the plane parameter.

write_mode: determines how the drawing pattern, specified by the most recent AGCMD call (or by aw_pattern for AWCMD), is used to modify bits in display memory.

Parameter choices are REPLACE, COMPLEMENT, RESET, or SET.

COMPLEMENT - drawing commands
RESET complement, reset,
SET or set, respectively, only those bits in display memory which correspond to ones in the drawing pattern.

REPLACE - the same as SET, except that drawing commands also reset the bits in display memory which correspond to zeros in the drawing pattern.

x1,y1: the beginning coordinates of a line, rectangle, rectangular area to be filled, or character.

x2,y2: the ending coordinates of a line.

xc,yc: the center coordinates of an arc or circle.

xres: the horizontal resolution, or the number of pixels displayed in the x direction.

The xres parameter is dependent on hs, hfp, and hbp. The following procedure for determining these four parameters makes the most efficient use of the bit map on the iSBX 275 board. AW is the number of active display words (xres÷16) per raster line. The GDC requires that AW be an even integer.

Interlaced Display Mode or Black-and-white, Non-interlaced Display Mode

In interlaced display mode (or in black-and-white, non-interlaced display mode), the horizontal timing parameters must satisfy the equation below (Eq. 1x). AW is an even integer, and the quotient on the right must be rounded down to the nearest integer:

$$hs + hfp + hbp + AW = \frac{\text{iSBX 275 board bandwidth}}{16 \times (\text{monitor horizontal rate})}$$

(Eq. 1x)

The first step is to find AW. The maximum possible resolution supported by the iSBX 275 board in interlaced display mode (or in black-and-white, non-interlaced display mode) is

$$\text{resolution} \leq 512 \times 512$$

Given the desired monitor aspect ratio, the bit map must be configured as a grid with the same x:y ratio, such that

$$\text{resolution} = x \times y, \text{ or}$$

$$512 \times 512 \geq (AW \times 16) \times \frac{AW \times 16}{\text{aspect ratio}}$$

(Eq. 2x)

Using Eq. 3x below, which is derived from Eq. 2x, calculate the maximum value for AW, rounding down to the nearest even integer:

$$AW \leq 32 \times \sqrt{\text{aspect ratio}}$$

(Eq. 3x)

The next step is to calculate the minimum values for hs, hfp, and hbp. (Refer to the respective parameter definitions.) Then adjust the values of AW, hs, hfp, and hbp, as required, to satisfy Eq. 1x above. Finally, solve for xres using Eq. 4x below:

$$xres = 16 \times AW$$

(Eq. 4x)

Color, Non-interlaced Display Mode

In color, non-interlaced display mode, the horizontal timing parameters must satisfy the equation below (Eq. 5x). AW is an even integer, and the quotient on the right must be rounded down to the nearest integer:

$$hs + hfp + hbp + AW = \frac{\text{iSBX 275 board bandwidth}}{16 \times (\text{monitor horizontal rate})}$$

(Eq. 5x)

The first step is to find AW. The maximum possible resolution supported by the iSBX 275 board in color, non-interlaced display mode is

$$\text{resolution} \leq 256 \times 256$$

Given the desired monitor aspect ratio, the bit map must be configured as a grid with the same x:y ratio, such that

$$\text{resolution} = x \times y, \text{ or}$$

$$256 \times 256 \geq (AW \times 16) \times \frac{AW \times 16}{\text{aspect ratio}}$$

(Eq. 6x)

Using Eq. 7x below, which is derived from Eq. 6x, calculate the maximum value for AW, rounding down to the nearest even integer:

$$AW \leq 16 \times \sqrt{\text{aspect ratio}}$$

(Eq. 7x)

The next step is to calculate the minimum values for hs, hfp, and hbp. (Refer to the respective parameter definitions.) Then adjust the values of AW, hs, hfp, and hbp, as required, to satisfy Eq. 5x above. Finally, solve for xres using Eq. 8x below:

$$xres = 16 \times AW$$

(Eq. 8x)

yres: the vertical resolution, or the number of lines displayed.

The yres parameter is dependent on vs, vfp, and vbp. The following procedure for determining these four parameters makes the most efficient use of the bit map on the iSBX 275 board. AL is the number of active display lines per field.

Interlaced Display Mode

In interlaced display mode, the vertical timing parameters must satisfy the equation below (Eq. 1y). AL is an even integer, and the quotient on the right must be rounded down to the nearest even integer:

$$vs + vfp + vbp + AL = \frac{\text{monitor horizontal rate}}{\text{monitor field rate}} \qquad \textit{(Eq. 1y)}$$

The first step is to find AL. The maximum possible resolution supported by the iSBX 275 board in interlaced display mode is

$$\text{resolution} \leq 512 \times 512$$

Only half of the active display lines are contained in each field in interlaced display mode, so given xres,

$$\text{resolution} = x \times (2 \times y), \text{ or}$$

$$512 \times 512 \geq \text{xres} \times (2 \times AL) \qquad \textit{(Eq. 2y)}$$

Using Eq. 3y below, which is derived from Eq. 2y, calculate the maximum value for AL, rounding down to the nearest even integer:

$$AL \leq \frac{512 \times 512}{2 \times \text{xres}} \qquad \textit{(Eq. 3y)}$$

The next step is to calculate the minimum values for vs, vfp, and vbp. (Refer to the respective parameter definitions.) Then adjust the values of AL, vs, vfp, and vbp, as required, to satisfy Eq. 1y above. Finally, solve for yres using Eq. 4y below:

$$\text{yres} = 2 \times AL \qquad \textit{(Eq. 4y)}$$

Color, Non-interlaced Display Mode

In color, non-interlaced display mode, the vertical timing parameters must satisfy the equation below (Eq. 5y), in which the quotient on the right must be rounded down to the nearest integer:

$$vs + vfp + vbp + AL = \frac{\text{monitor horizontal rate}}{\text{monitor field rate}} \qquad \textit{(Eq. 5y)}$$

The first step is to find AL. The maximum possible resolution supported by the iSBX 275 board in color, non-interlaced display mode is

$$\text{resolution} \leq 256 \times 256$$

All of the active display lines are contained in each field in non-interlaced display mode, so given xres,

$$\text{resolution} = x \times y, \text{ or}$$

$$256 \times 256 \geq \text{xres} \times AL \qquad \textit{(Eq. 6y)}$$

Using Eq. 7y below, which is derived from Eq. 6y, calculate the maximum value for AL, rounding down to the nearest integer:

$$AL \leq \frac{256 \times 256}{\text{xres}} \qquad \textit{(Eq. 7y)}$$

The next step is to calculate the minimum values for vs, vfp, and vbp. (Refer to the respective parameter definitions.) Then adjust the values of AL, vs, vfp, and vbp, as required, to satisfy Eq. 5y above. Finally, solve for yres using Eq. 8y below:

$$\text{yres} = AL \qquad \textit{(Eq. 8y)}$$

Black-and-white, Non-interlaced Display Mode

In black-and-white, non-interlaced display mode, the vertical timing parameters must satisfy the equation below (Eq. 9y), in which the quotient on the right must be rounded down to the nearest integer:

$$vs + vfp + vbp + AL = \frac{\text{monitor horizontal rate}}{\text{monitor field rate}} \qquad \textit{(Eq. 9y)}$$

The first step is to find AL. The maximum possible resolution supported by the iSBX 275 board in black-and-white, non-interlaced display mode is

$$\text{resolution} \leq 512 \times 512$$

All of the active display lines are contained in each field in non-interlaced display mode, so given xres,

$$\text{resolution} = x \times y, \text{ or}$$

$$512 \times 512 \geq \text{xres} \times \text{AL} \qquad \textit{(Eq. 10y)}$$

Using Eq. 11y below, which is derived from Eq. 10y, calculate the maximum value for AL, rounding down to the nearest integer:

$$\text{AL} \leq \frac{512 \times 512}{\text{xres}} \qquad \textit{(Eq. 11y)}$$

The next step is to calculate the minimum values for vs, vfp, and vbp. (Refer to the respective parameter definitions.) Then adjust the values of AL, vs, vfp, and vbp, as required, to satisfy Eq. 9y above. Finally, solve for yres using Eq. 12y below:

$$\text{yres} = \text{AL} \qquad \textit{(Eq. 12y)}$$

zoom_factor: the factor by which a specified figure or character is magnified at the time it is drawn in the bit map.

Parameter choices are ONE_X, TWO_X, THREE_X, FOUR_X, FIVE_X, SIX_X, SEVEN_X, EIGHT_X, NINE_X, TEN_X, ELEVEN_X, TWELVE_X, THIRTEEN_X, FOURTEEN_X, FIFTEEN_X, or SIXTEEN_X.

APPLICATION EXAMPLES

Introduction

The application examples described below were developed and run on a SYSTEM 86/330A, which is an iRMX 86 based system containing the following:

- an iSBC 86/30 processor board with 128KB of onboard RAM
- an iSBC 337 Numeric Data Processor MULTIMODULE board, mounted on the iSBC 86/30 board
- an iSBC 56A 256KB RAM board
- an iSBC 215 Winchester disk controller board
- an iSBX 218 flexible disk controller MULTIMODULE board, mounted on the iSBC 215 board
- a 35MB Winchester disk drive
- an 8" flexible disk drive

The system ran the pre-configured version of iRMX 86, Rel. 5, and the application examples were written in PL/M-86.

The iSBX 275 board was connected to SBX port J4 in the center of the iSBC 86/30 board. A Mitsubishi C-3419 RGB color monitor was used for the display.

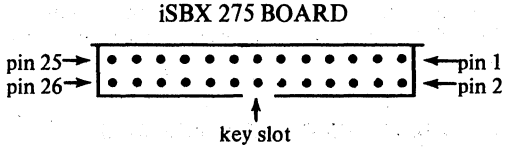
Overview

First, instructions are provided for building a cable to interface the iSBX 275 board to an RGB color monitor. Next, the AIRPLANE application is discussed, providing an example of black-and-white, interlaced display mode operation. Finally, the REACTOR application is discussed, providing an example of color display mode operation.

iSBX™ 275 Board Color Monitor Cable Fabrication

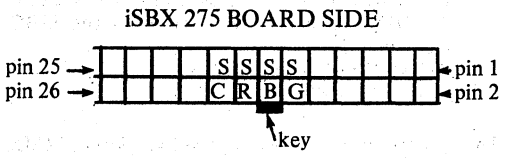
Since a BNC color monitor was chosen, a female BNC connector, Amphenol part number 69475, was first attached to each of four six-foot lengths of 75 ohm coax cable, Belden part number 83264-001. The four connectors were then labeled Red, Green, Blue, and Csync, corresponding to the respective male BNC connectors on the back of the monitor. Two crimp-on connector pins, BERG part number 75691-014, were then crimped and soldered to the other end of each cable piece, one to the wire and the other to the shielding. Finally, the four cable pieces were attached to a wire-type, keyed, 26-pin female connector, BERG part number 65846-819, as described below.

The keyed, 26-pin male connector on the iSBX 275 board is depicted here as viewed from above.



The 26-pin female connector for the cable is also depicted here as viewed from above (when inserted on the board connector). The crimp pins on the four cable pieces were inserted into this connector from the top. The figure shows the locations for inserting

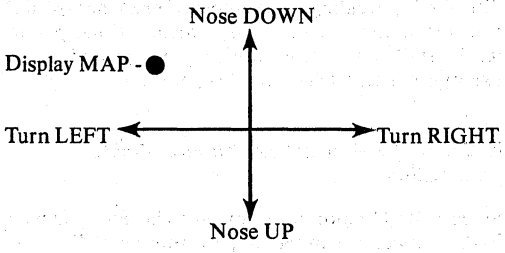
all eight of the Red(R), Green(G), Blue(B), Csync(C), and Shielding(S) crimp pins.



Airplane Application

INTRODUCTION

This application example is a simulation of a Cessna 152. For this example, the iSBX 275 board is configured for black-and-white, interlaced display mode operation. An Atari joystick, oriented such that the button is in the upper left corner, controls the "airplane" as shown.



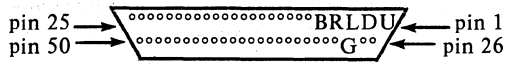
A real-time "flight panel" on the graphics monitor (see Fig. 9) reflects the "movement" of the plane as it is input by the joystick. When the joystick button is depressed and held, a map replaces the flight panel on the monitor. This map depicts the ground track of the plane and the surrounding area.

OVERVIEW

First, instructions are provided for building a cable to interface an Atari joystick to the iSBC 86/30 board. Then the necessary jumpers on the iSBC 86/30 board and the iSBX 275 board are detailed. Next, the process is described for determining the parameters for the INIT routine. Finally, the generation of the flight panel display shown in Fig. 9 is discussed.

ATARI JOYSTICK CABLE FABRICATION

A cable was built to connect an Atari joystick to the printer port 50-pin connector on the back of the SYSTEM 86/330A chassis. This enabled the joystick to communicate with the system via the parallel port on the iSBC 86/30 board. First, six three-foot lengths of 22 AWG wire were soldered to a 50-pin male connector, Ansley part number PN 609-50M. The figure below shows the locations for soldering the six wires to the connector, depicted here as viewed from the wire side. The letters represent the functions (U)p, (D)own, (L)eft, (R)ight, (B)utton, and (G)round.



Then the six wires were soldered to a 9-pin male connector. The figure below shows the locations for soldering the wires to the connector, depicted here as viewed from the wire side.

JUMPERS

A jumper was installed on the iSBC 86/30 board between posts 44 and 53. This made Port C, bit 7, available over the parallel port of the board.

This application required the maximum available resolution of the iSBX 275 board. The board was therefore jumpered for black-and-white display mode operation, resulting in a nominal resolution of 512 × 512. The board was also jumpered to provide an active low combined sync (CSYNC/) signal for the color monitor. The black-and-white display mode was selected by removing jumper 12 - 13. All of the installed jumpers are described below.

1 - 2 test jumper
8 - 10 provides CSYNC/ output
14 - 15 test jumper
16 - 17 selects non-divided clock

GRAPHICS INITIALIZATION

Introduction

In order to use the graphics library of PL/M-86 routines described earlier, the following include file was placed in the PLANE module:

```
$include(/user/sbx275/ext/sbx275)
```

An application program must call the procedure INIT before it calls any other procedure from this graphics library. Typically, INIT is called in the main procedure of an application, as was done in PLANE. (Refer to Fig. 10.) The iSBX 275 board is initialized according to the parameters accompanying the INIT

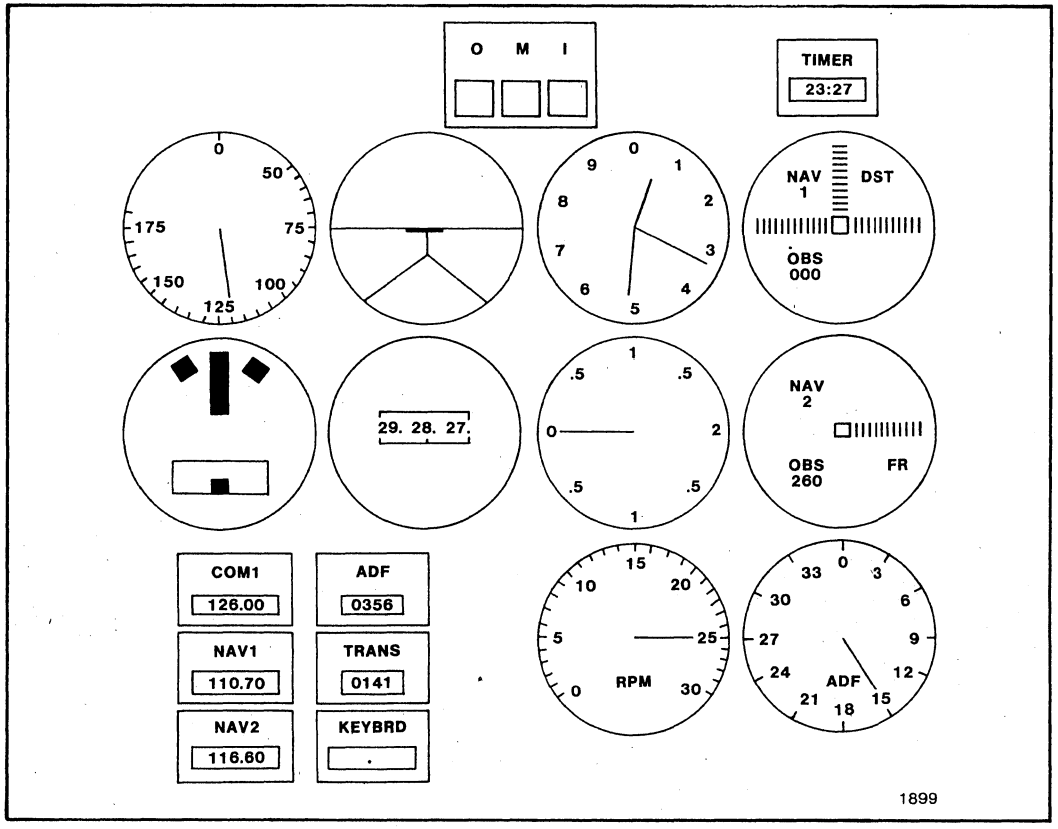


Figure 9. Flight Panel Display from Plane

call. These parameters specify the desired resolution, monitor, and mode of operation. They also specify the I/O port addresses of the iSBX 275 board. Following the INIT call, an application program may begin drawing by calling any of the other graphics procedures.

Overview

Many steps are involved in determining the parameters for INIT. First, the number and size of the bit map planes must be specified, providing the nominal resolution. Then the exact horizontal resolution and the associated monitor timing parameters must be determined. Next, the vertical resolution and the associated monitor timing parameters must be determined. Finally, the mode of operation and I/O addresses of the iSBX 275 board must be specified. The INIT parameters for PLANE were determined as described below.

Color Plane Size

The airplane application requires the black-and-white display mode of the iSBX 275 board. Referring to the description of the plane_size parameter, we find for this mode that

plane_size = 16384 (or 4000h).

Horizontal Resolution

The airplane application requires the interlaced display mode of the monitor. Referring to the description of the xres parameter, we find that the horizontal timing parameters must satisfy Eq. 1x. Given the 12.6 MHz oscillator shipped on the iSBX 275 board and the monitor horizontal rate of 15.75 KHz, Eq. 1x becomes

$$hs + hfp + hbp + AW = \frac{12.6\ \text{MHz}}{16 \times 15.75\ \text{KHz}}$$

$$= 50, \qquad \textit{(Eq. P1)}$$

which is an integer value as required.

```
*************************************************
                    MAIN
*************************************************

 1219   test: procedure public;

 1220   declare (x1,x2,y1,y2) word;

 1221   declare adc$use byte;
 1222   declare mkr$test byte;
 1223   declare trim real;
 1224   declare (low$byte, high$byte) word;

        /* initialize controller */
 1225   call init (16384,576,452,5,4,4,5,16,16,
                                  1bh,82h,80h);
        /* initial frequencies */
 1226   call movb (@initial$com(0),
                                  @com$freq(0),5);
 1227   call movb (@initial$nav1(0),
                                  @nav1$freq(0),5);
 1228   call movb (@initial$nav2(0),
                                  @nav2$freq(0),5);
 1229   call movb (@initial$adf(0),
                                  @adf$freq(0),4);
 1230   call movb (@initial$tran(0),
                                  @tran$freq(0),4);
```

Figure 10. INIT Call In PLANE

Still referring to the xres parameter description, we find that the maximum value for AW is determined using Eq. 3x. A 4:3 aspect ratio is desired, so given the nominal 255mm:190mm aspect ratio of the monitor, Eq. 3x becomes

$$AW \leqslant 32 \times \sqrt{255\text{mm} \div 90\text{mm}} = 37.1,$$

which, when rounded down as required to the nearest even integer, becomes

$$AW \leqslant 36.$$

The monitor manual specifies a minimum horizontal sync of 4.4 μs. Referring to the description of the HS parameter, we find that the minimum value for hs must be the greater of '5' and

$$hs \geqslant 4.4\ \mu s \times 12.6\ \text{MHz} \div 16 = 3.47,$$

and since $5 > 3.47$,

$$hs \geqslant 5.$$

The monitor manual specifies a minimum horizontal front porch of 1.2μs. Referring to the description of the hfp parameter, we find that the minimum value for hfp must be the greater of '2' and

$$hfp \geqslant 1.2\ \mu s \times 12.6\ \text{MHz} \div 16 = 0.945,$$

and since $2 > 0.945$,

$$hfp \geqslant 2.$$

The monitor manual specifies a minimum horizontal back porch of 4.9 μs. Referring to the description of the hbp parameter, we find that the minimum value for hbp must be the greater of '5' and

$$hbp \geqslant 4.9\ \mu s \times 12.6\ \text{MHz} \div 16 = 3.86,$$

and since $5 > 3.86$,

$$hbp \geqslant 5.$$

Using the values determined above,

$$hs + hfp + hbp + AW = 5 + 2 + 5 + 36 = 48,$$

which is two less than required by Eq. P1 above. The hs, hfp, and hbp values determined above are minimums, while the AW value is a maximum. Since the sum of these values must be incremented by two to satisfy Eq. P1, the hfp parameter is arbitrarily selected to be incremented by two. The horizontal parameter values are thus

$$AW = 36,\ hs = 5,\ hfp = 4,\ hbp = 5.$$

Referring once more to the description of the xres parameter, we find from Eq. 4x that

$$xres = 16 \times 36 = 576.$$

Vertical Resolution

Referring to the description of the yres parameter, we find that the vertical timing parameters must satisfy Eq. 1y. Given the monitor field rate of 60 Hz, Eq. 1y becomes

$$vs + vfp + vbp + AL = 15.75\ \text{KHz} \div 60\ \text{Hz} = 262.5,$$

which, when rounded down as required to the nearest even integer, becomes

$$vs + vfp + vbp + AL = 262. \qquad \textit{(Eq. P2)}$$

Still referring to the yres parameter description, we find that the maximum value for AL is determined using Eq. 3y, which becomes

$$AL \leqslant (512 \times 512) \div (2 \times 576) = 227.6,$$

which, when rounded down as required to the nearest even integer, becomes

$$AL \leqslant 226.$$

The monitor manual specifies a minimum vertical sync of three horizontal retrace-and-scan periods. Referring to the description of the vs parameter, we find that the minimum value for vs must be

$$vs \geq (3 \div 15.75\,KHz) \times 15.75\,KHz = 3.$$

The monitor manual specifies a minimum vertical front porch of three horizontal retrace-and-scan periods. Referring to the description of the vfp parameter, we find that the minimum value for vfp must be

$$vfp \geq (3 \div 15.75\,KHz) \times 15.75\,KHz = 3.$$

The monitor manual specifies a minimum vertical back porch of twelve horizontal retrace-and-scan periods. Referring to the description of the vbp parameter, we find that the minimum value for vbp must be

$$vbp \geq (12 \div 15.75\,KHz) \times 15.75\,KHz = 12.$$

Using the values determined above,

$$vs+vfp+vbp+AL=3+3+12+226=244,$$

which is eighteen less than required by Eq. P2 above. The vs, vfp, and vbp values determined above are minimums, while the AL value is a maximum. Since the sum of these values must be incremented by eighteen to satisfy Eq. P2, the vs, vfp, and vbp parameters are arbitrarily selected to be incremented by 1, 13, and 4, respectively. The vertical parameter values are thus

$$AL = 226,\ vs = 4,\ vfp = 16,\ vbp = 16.$$

Referring once more to the description of the yres parameter, we find from Eq. 4y that

$$yres = 2 \times 226 = 452.$$

Operation Mode and I/O Addresses

Referring to the description of the opmode parameter, we choose

opmode = 1BH.

This initializes the GDC for interlaced display mode operation; for individual pixel addressing and drawing; and for drawing in display memory only during horizontal and vertical retrace, when display memory is blanked from the monitor.

The iSBX 275 board resides on SBX port J4 of the iSBC 86/30 board. Referring to the description of the command_port and data_port parameters, we find that the I/O address chosen for each of these parameters must activate the MCS0/ signal on the SBX bus. Referring then to the iSBC 86/14 AND iSBC 86/30 SINGLE BOARD COMPUTER HARDWARE REFERENCE MANUAL, Intel order number 144044-002, page 3-7, we find that any of the I/O addresses 80, 82, 84, 86, 88, 8A, 8C, or 8E will activate MCS0/. Finally, referring to the iSBX 275 VIDEO GRAPHICS CONTROLLER MULTIMODULE BOARD REFERENCE MANUAL, Intel order number 144829-001, page 3-1, Table 3-1, we find that any of the I/O addresses 82, 86, 8A, or 8E may be used for the command_port parameter, and any of the I/O addresses 80, 84, 88, or 8C may be used for the data_port parameter. The I/O addresses chosen are therefore

command_port = 82H,

data_port = 80H.

CODE GENERATION

PLANE generates two different bit map displays, one for the flight panel and one for the ground track. Based on inputs from a joystick, these displays are updated in the bit map and displayed on the monitor screen. This section addresses only the flight panel display.

Before any code was generated, the flight panel was drawn on grid paper. This paper was layed out according to the xres and yres parameter values determined above for the INIT routine. The paper had 576 squares in the horizontal direction and 452 squares in the vertical direction, so each square represented a pixel on the display. The lower left square was assigned the (x,y) coordinates (0,0), with x increasing to the right and y increasing up.

The DRAW$PANEL procedure was then generated. This procedure draws the various flight panel elements in display memory, according to their size and location on the grid paper. (Refer to Fig. 9.) DRAW$PANEL first draws the six control boxes in the lower left corner of the flight panel display. Then the marker and timer boxes at the top of the flight panel display are drawn. Finally, the stationary portions of the ten indicators are drawn. The airspeed indicator in the upper left corner is drawn first. Then the turn-and-bank indicator, directly below the airspeed indicator, is drawn. Next, the artificial horizon indicator, directly to the right of the airspeed indicator, is drawn. The remaining seven indicators are drawn in the same fashion, progressing from top to bottom and from left to right. The dynamic portions of the flight panel display are drawn by the update procedures for the various controls and indicators.

Since PLANE generates two different bit map displays, DRAW$PANEL first clears display memory. (Refer to the listing of DRAW$PANEL in Appendix A.1.) It does this by calling UPCMD and AWCMD as explained in the AWCMD description. Then, since the drawing pattern is in an unknown state at the invocation of DRAW$PANEL, AGCMD is called to set the drawing pattern for solid line drawing and area fill. All lines subsequently drawn will thus be solid, and all areas filled will be solid filled. Next, UMCMD is called to change the write mode from REPLACE, which is the default value set by INIT, to SET. For each word written to the bit map in SET mode, only bit map locations corresponding to logical '1' bits in the word are set to '1'. Bit map locations corresponding to logical '0' bits in the word remain unchanged. The various elements of the flight panel are then drawn using UBCMD, UCCMD, ULCMD, UFCMD, and STRCMD. As pointed out in the STRCMD description, STRCMD changes the drawing pattern. Therefore, following each series of STRCMD calls and before resuming drawing, AGCMD is called to reload the drawing pattern.

Procedures were then generated which, based on input from a joystick, update the various controls and indicators in the flight panel display. One of these procedures, ALTIMETER, updates the three hands on the altimeter indicator. (Refer to the listing of ALTIMETER in Appendix A.2.) First, an AGCMD call establishes solid line drawing, since the hands were drawn as solid lines. Next, a UMCMD call sets the drawing mode to CLEAR. For each word written to the bit map in CLEAR mode, only bit map locations corresponding to logical '1' bits in the word are set to '0'. Bit map locations corresponding to logical '0' bits in the word remain unchanged. The old hands are then redrawn using ULCMD, with the result that they are erased. A second UMCMD call changes the drawing mode to SET, after which the new hands are drawn using ULCMD. The rest of the flight panel display is updated in similar fashion.

Reactor Application

INTRODUCTION

This application example is a non-interactive simulation of a reactor (see Fig. 11). The fluids in the various closed systems comprising the reactor move in simulated flow. For this application, the iSBX 275 board is configured for color mode operation.

OVERVIEW

First, the necessary jumpers on the iSBX 275 board are detailed. Next, the process is described for determining the parameters for the INIT routine. Finally, the generation of the reactor display shown in Fig. 11 is discussed.

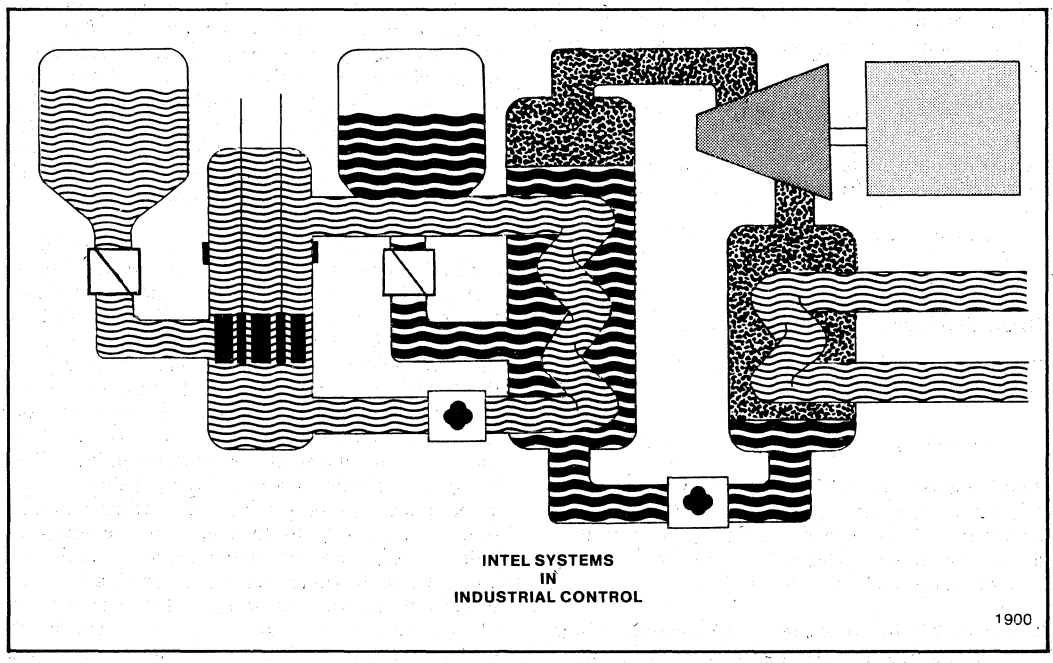


Figure 11. Reactor Display

JUMPERS

This application required the color capabilities of the iSBX 275 board. The board was therefore jumpered for color mode operation, resulting in a nominal resolution of 256 × 256. The board was also jumpered to provide an active low combined sync (CSYNC/) signal for the color monitor. All of the installed jumpers are described below.

1 - 2 test jumper
8 - 10 provides CSYNC/ output
12 - 13 selects color display mode
14 - 15 test jumper
17 - 18 selects divided-by-two mode

GRAPHICS INITIALIZATION

Introduction

In order to use the graphics library of PL/M-86 routines described earlier, the following include file was placed in the REACTOR module:

```
$include(/user/sbx275/ext/sbx275)
```

An application program must call the procedure INIT before it calls any other procedure from this graphics library. Typically, INIT is called in the main procedure of an application, as was done in REACTOR. (Refer to Fig. 12.) The iSBX 275 board is initialized according to the parameters accompanying the INIT call. These parameters specify the desired resolution, monitor, and mode of operation. They also specify the I/O port addresses of the iSBX 275 board. Following the INIT call, an application program may begin drawing by calling any of the other graphics procedures.

```
*************************************************
                    MAIN
*************************************************

  925   call init(4096,288,227,2,3,2,3,16,16,
                                  12h,82h,80h);
  926   call reactor;
  927   call dqexit(0);

  928   end reactor$module;
```

Figure 12. INIT Call In REACTOR

Overview

Many steps are involved in determining the parameters for INIT. First, the number and size of the bit map planes must be specified, providing the nominal resolution. Then the exact horizontal resolution and the associated monitor timing parameters must be determined. Next, the vertical resolution and the associated monitor timing parameters must be determined. Finally, the mode of operation and I/O addresses of the iSBX 275 board must be specified. The INIT parameters for REACTOR were determined as described below.

Color Plane Size

The reactor application requires the color display mode of the iSBX 275 board. Referring to the description of the plane_size parameter, we find for this mode that

plane_size = 4096 (or 1000h).

Horizontal Resolution

The reactor application requires the non-interlaced display mode of the monitor. Referring to the description of the xres parameter, we find that the horizontal timing parameters must satisfy Eq. 5x. Given the 12.6 MHz oscillator shipped on the iSBX 275 board, the divide-by-two mode, and the monitor horizontal rate of 15.75 KHz, Eq. 5x becomes

$$hs + hfp + hbp + AW = \frac{6.3\ \text{MHz}}{16 \times 15.75\ \text{KHz}} = 25, \qquad \textit{(Eq. P1)}$$

which is an integer value as required.

Still referring to the xres parameter description, we find that the maximum value for AW is determined using Eq. 7x. A 4:3 aspect ratio is desired, so given the nominal 255mm:190mm aspect ratio of the monitor, Eq. 7x becomes

$$AW \leqslant 16 \times \sqrt{255\text{mm} \div 90\text{mm}} = 18.54,$$

which, when rounded down as required to the nearest even integer, becomes

$$AW \leqslant 18.$$

The monitor manual specifies a minimum horizontal sync of 4.4 μs. Referring to the description of the hs parameter, we find that the minimum value for hs must be

$$hs \geqslant 4.4\ \mu s \times 6.3\ \text{MHz} \div 16 = 1.733,$$

which, when rounded up as required to the nearest integer, becomes

$$hs \geqslant 2.$$

The monitor manual specifies a minimum horizontal front porch of 1.2 μs. Referring to the description of the hfp parameter, we find that the minimum value for hfp must be the greater of '2' and

$$hfp \geq 1.2\ \mu s \times 6.3\ MHz \div 16 = 0.473,$$

and since $2 > 0.473$,

$$hfp \geq 2.$$

The monitor manual specifies a minimum horizontal back porch of 4.9 μs. Referring to the description of the hbp parameter, we find that the minimum value for hbp must be the greater of '3' and

$$hbp = 4.9\ \mu s \times 6.3\ MHz \div 16 = 1.929,$$

and since $3 > 1.929$,

$$hbp \geq 3.$$

Using the values determined above,

$$hs+hfp+hbp+AW=2+2+3+18=25,$$

satisfying Eq. P1 above. The horizontal parameter values are thus

$$AW = 18,\ hs = 2,\ hfp = 2,\ hbp = 3.$$

Referring once more to the description of the xres parameter, we find from Eq. 8x that

$$xres = 16 \times 18 = 288.$$

Vertical Resolution

Referring to the description of the yres parameter, we find that the vertical timing parameters must satisfy Eq. 5y. Given the monitor field rate of 60 Hz, Eq. 5y becomes

$$vs+vfp+vbp+AL=15.75\ KHz \div 60\ Hz=262.5,$$

which, when rounded down as required to the nearest integer, becomes

$$vs + vfp + vbp + AL = 262. \qquad \textit{(Eq. P2)}$$

Still referring to the yres parameter description, we find that the maximum value for AL is determined using Eq. 7y1, which becomes

$$AL \leq (256 \times 256) \div 288 = 227.6,$$

which, when rounded down as required to the nearest integer, becomes

$$AL \leq 227.$$

The monitor manual specifies a minimum vertical sync of three horizontal retrace-and-scan periods. Referring to the description of the vs parameter, we find that the minimum value for vs must be

$$vs \geq (3 \div 15.75\ KHz) \times 15.75\ KHz = 3.$$

The monitor manual specifies a minimum vertical front porch of three horizontal retrace-and-scan periods. Referring to the description of the vfp parameter, we find that the minimum value for vfp must be

$$vfp \geq (3 \div 15.75\ KHz) \times 15.75\ KHz = 3.$$

The monitor manual specifies a minimum vertical back porch of twelve horizontal retrace-and-scan periods. Referring to the description of the vbp parameter, we find that the minimum value for vbp must be

$$vbp \geq (12 \div 15.75\ KHz) \times 15.75\ KHz = 12.$$

Using the values determined above,

$$vs+vfp+vbp+AL=3+3+12+227=245,$$

which is seventeen less than required by Eq. P2 above. The vs, vfp, and vbp values determined above are minimums, while the AL value is a maximum. Since the sum of these values must be incremented by seventeen to satisfy Eq. P2, the vfp and vbp parameters are arbitrarily selected to be incremented by 13 and 4, respectively. The vertical parameter values are thus

$$AL = 227,\ vs = 3,\ vfp = 16,\ vbp = 16.$$

Referring once more to the description of the yres parameter, we find from Eq. 8y that

$$yres = 227.$$

Operation Mode and I/O Addresses

Referring to the description of the opmode parameter, we choose

$$opmode = 12H.$$

This initializes the GDC for non-interlaced display mode operation; for individual pixel addressing and drawing; and for drawing in display memory only during horizontal and vertical retrace, when display memory is blanked from the monitor.

The iSBX 275 board resides on SBX port J4 of the iSBC 86/30 board. Referring to the description of the command_port and data_port parameters, we find that the I/O address chosen for each of these

parameters must activate the MCS0/ signal on the SBX bus. Referring then to the iSBC 86/14 AND iSBC 86/30 SINGLE BOARD COMPUTER HARDWARE REFERENCE MANUAL, Intel order number 144044-002, page 3-7, we find that any of the I/O addresses 80, 82, 84, 86, 88, 8A, 8C, or 8E will activate MCS0/. Finally, referring to the iSBX 275 VIDEO GRAPHICS CONTROLLER MULTIMODULE BOARD REFERENCE MANUAL, Intel order number 144829-001, page 3-1, Table 3-1, we find that any of the I/O addresses 82, 86, 8A, or 8E may be used for the command_port parameter, and any of the I/O addresses 80, 84, 88, or 8C may be used for the data_port parameter. The I/O addresses chosen are therefore

command_port = 82H,

data_port = 80H.

CODE GENERATION

This section addresses the generation of the REACTOR bit map display. To begin with, before any code was generated, the reactor was drawn on grid paper. This paper was layed out according to the xres and yres parameter values determined above for the INIT routine. The paper had 288 squares in the horizontal direction and 227 squares in the vertical direction, so each square represented a pixel on the display. The lower left square was assigned the (x,y) coordinates (0,0), with x increasing to the right and y increasing up.

The code was then generated to draw the various reactor display elements in the bit map, according to their size and location on the grid paper. (Refer to Fig. 11.) This code calls UMCMD each time the color needs to be changed, and it draws the entire reactor display using ULCMD and UFCMD. It takes advantage of the fact that INIT clears display memory, initializes the drawing pattern for solid line drawing and area filling, and initializes the write mode to SET. For each word written to the bit map in SET mode, only bit map locations corresponding to logical '1' bits in the word are set to '1'. Bit map locations corresponding to logical '0' bits in the word remain unchanged.

REACTOR first sets the drawing color to WHITE. (Refer to the listing of REACTOR in Appendix B.) The outlines of the reactor vessel, the steam generator, and the condenser pipe are then drawn, and the turbine-to-generator shaft is filled in. Then the drawing color is changed to RED and the turbine is filled in. Next, the drawing color is changed to GREEN and the generator, the valve blocks, and the pump blocks are filled in. The drawing color is next changed to YELLOW. The valves and gears are then filled in over the green blocks, with the result that the valves and gears appear yellow. Then the drawing color is changed to MAGENTA, and the stationary reactor elements are filled in. Next, the drawing color is changed to CYAN and the moveable reactor elements are filled in. Finally, the drawing color is changed to BLUE. The fluids in the system are then filled in, followed by the steam in the steam generator.

Code was then generated to simulate motion in the fluids and steam. First, a UMCMD call changes the write mode from SET to COMPLEMENT. For each word written to the bit map in COMPLEMENT mode, only bit map locations corresponding to logical '1' bits in the word are complemented. Bit map locations corresponding to logical '0' bits in the word remain unchanged. For motion in the fluids, an AGCMD call establishes a dotted line drawing pattern. Every second line is then repeatedly rewritten, in sequence. This simulates motion by turning half of the pixels in these lines alternately off and on. Motion in the steam is simulated by repeatedly writing every second line twice, in sequence. First, an AGCMD call establishes the old line drawing pattern for a line. Then the line is redrawn, with the result that it is erased. A second AGCMD call establishes a new line drawing pattern for the line, and the new line is drawn.

PERFORMANCE CALCULATIONS

Introduction

The iSBX 275 board applications discussed in this application note have attempted to maximize display memory utilization and hence resolution. Since maximizing resolution may have the opposite effect on performance, this section discusses some of the performance and resolution tradeoffs which pertain to the iSBX 275 board.

For this discussion, performance refers to the speed with which an image can be updated in display memory. Typically, drawing in display memory is allowed only during retrace blanking, as in the examples described in this application note. The GDC can be initialized to allow drawing in display memory at any time (refer to the definition of the opmode parameter), but the resulting display flashing generally is not tolerable. Display flashing occurs when data is written to bit map locations which are being accessed for monitor refresh. Since a monitor screen is refreshed only during active display time, it is assumed here that the iSBX 275 board is always operated in flashless mode, such that drawing in display memory only occurs during retrace blanking.

The minimum bandwidth of the iSBX 275 board is 4 MHz. The maximum bandwidth is 10 MHz in color display mode and 13 MHz in black-and-white display

mode. In the divide-by-two mode, in which the oscillator frequency is divided by two, a 20 MHz oscillator may be used in color display mode, or a 25 MHz oscillator may be used in black-and-white display mode. The following performance discussion assumes that the 12.6 MHz oscillator shipped on the iSBX 275 board is used for all modes. It should be noted that for color display mode, performance can be nearly doubled by changing from the default oscillator (6.3 MHz bandwidth) to an oscillator which provides the full 10 MHz bandwidth available.

Overview

For the purpose of this discussion, a nominal 512 × 512 black-and-white image is the standard for comparison. Each pixel of this image is represented by one bit in display memory. The choice of color display mode limits the nominal resolution to 256 × 256 pixels. With one-fourth as many pixels, one might think that a color image would provide four times the performance of a 512 × 512 black-and-white image. However, each pixel of a color image is represented by three bits in display memory, one in each of three color planes. The primary colors are generated by drawing in one plane, the comlementary colors by drawing in two planes, and white is generated by drawing in all three planes. Since a color image typically contains a mixture of colors, it requires on the average that two bits be drawn in display memory for each pixel. This would imply that a color image provides twice the performance of a 512 × 512 black-and-white image. Since the default oscillator frequency must be divided in half in color display mode, though, there is essentially no difference in the drawing performance for a 256 × 256 8-color image and 512 × 512 black-and-white image.

If the default oscillator restriction is removed, a 256 × 256 8-color image can provide up to 60% better performance than a 512 × 512 black-and-white image. This is accomplished by replacing the default 12.6 MHz oscillator with a 20 MHz oscillator, with the iSBX 275 board in the divide-by-two mode; or by replacing the default oscillator with a 10 MHz oscillator, with the board in the non-divided mode. This increases the bandwidth of the board from 6.3 MHz to 10 MHz.

If a nominal resolution of 256 × 256 pixels is sufficient, a 256 × 256 black-and-white image can provide four times the performance of a 512 × 512 black-and-white image. This is due to the fact that the 512 × 512 image has four times as many pixels as the 256 × 256 image.

With drawing in display memory taking place only during retrace blanking, performance improvements can be realized by increasing the blanking time. Increasing the vertical blanking time decreases the number of display lines, and increasing the horizontal blanking time decreases the number of active words per line. Therefore, in order to maximize performance, the desired image should be drawn with the minimum allowable resolution. This minimizes the number of active display words comprising the image, thereby minimizing the time required to update the image in the bit map.

Available Drawing Time

The following equation expresses the fraction of each raster-scan cycle which is available for drawing in display memory. This fraction represents the theoretical limit on the drawing time available, enabling the calculation of the maximum theoretical performance of the iSBX 275 board. The larger the fraction, the higher the theoretical maximum performance, with a fully blanked display (which obviously has no intrinsic value) yielding a value of one. The actual performance is dependent on external factors, such as the amount of processing time which the CPU requires in order to generate the data for updating an image. For the equation below, the horizontal blanking time (hs + hfp + hbp) must be rounded down to the nearest even number. It is expressed in display words, as two display word times are necessary to draw each bit in display memory.

$$\text{FRACTION} = \frac{\left(\frac{hs+hfp+hbp}{hs+hfp+hbp+AW} \times AL\right) + (vs+vfp+vbp)}{vs+vfp+vbp+AL}$$

where

AW = number of active words,
AL = number of active lines,
hs + hfp + hbp = horizontal blanking time,
hs + hfp + hbp + AW = total horizontal time,
vs + vfp + vbp = vertical blanking time,
vs + vfp + vbp + AL = total vertical time.

As a general rule of thumb, taking into account external factors, the actual performance of the iSBX 275 board is approximated using

FRACTION = .25

The equation below expresses the performance of the iSBX 275 board in pixels per second, assuming graphics mode operation. Thirty-two oscillator cycles are consumed for each bit drawn in display memory.

$$\text{PERFORMANCE} = \frac{\text{FRACTION} \times (\text{iSBX 275 board bandwidth})}{32 \times (\text{number of color planes used})}$$

Plane Example

For the PLANE application discussed previously, the fraction of each raster-scan cycle available for drawing in display memory is

$$\text{FRACTION} = \frac{\left(\frac{5+4+5}{5+4+5+36} \times 226\right) + (4+16+16)}{4+16+16+226}$$

$$= \frac{(14 \times 226 \div 50) + 36}{262} = .379$$

The theoretical maximum performance of the iSBX 275 board for PLANE is therefore

$$\text{PERFORMANCE} = .379 \times (12.6\,\text{MHz}) \div 32$$

$$= 149{,}200 \text{ pixels per second,}$$

and the actual performance is approximated by

$$\text{PERFORMANCE} = .25 \times (12.6\,\text{MHz}) \div 32$$

$$= 98{,}400 \text{ pixels per second.}$$

Reactor Example

For the REACTOR application discussed previously, the fraction of each raster-scan cycle available for drawing in display memory is

$$\text{FRACTION} = \frac{\left(\frac{2+2+3}{2+2+3+18} \times 227\right) + (3+16+16)}{3+16+16+227}$$

$$= \frac{(7 \times 227 \div 25) + 35}{262} = .376$$

Assuming that all three color planes were written to for each color, which would happen only if everything were drawn in white, the theoretical maximum performance of the iSBX 275 board for REACTOR would be

$$\text{PERFORMANCE} = \frac{.376 \times (12.6\,\text{MHz} \div 2)}{32 \times 3}$$

$$= 24{,}600 \text{ pixels per second.}$$

Assuming that only one color plane were written to for each color, which would happen only if everything were drawn in primary colors, the theoretical maximum performance of the iSBX 275 board for REACTOR would be

$$\text{PERFORMANCE} = \frac{.376 \times (12.6\,\text{MHz} \div 2)}{32}$$

$$= 74{,}000 \text{ pixels per second.}$$

Since all seven available colors are used, each color drawn in display memory requires writing to an average of two color planes. The theoretical maximum performance of the iSBX 275 board for REACTOR is therefore best represented by

$$\text{PERFORMANCE} = \frac{.376 \times (12.6\,\text{MHz} \div 2)}{32 \times 2}$$

$$= 37{,}000 \text{ pixels per second,}$$

and the actual performance is approximated by

$$\text{PERFORMANCE} = \frac{.25 \times (12.6\,\text{MHz} \div 2)}{32 \times 2}$$

$$= 24{,}600 \text{ pixels per second.}$$

Image Updating

From the performance numbers derived above, relative display image update times can be calculated. The actual number of pixels involved in an image update is application dependent. Therefore, for this discussion, it will be assumed that a typical image update involves half of the pixels displayed. The following equation represents the time required to update a display image.

$$\text{UPDATE} = \frac{\text{number of pixels updated}}{\text{PERFORMANCE}}$$

For a black-and-white display image, the maximum possible resolution is

$$\text{RESOLUTION} = 512 \times 512 = 262{,}144 \text{ pixels.}$$

Therefore, using (RESOLUTION ÷ 2) as the number of pixels updated, the time required to update a 512 × 512 resolution black-and-white display image is approximated by

$$\text{UPDATE} = \frac{262{,}144 \div 2}{98{,}400} = 1.332 \text{ seconds.}$$

For a color display image, the maximum possible resolution is

$$\text{RESOLUTION} = 256 \times 256 = 65{,}536 \text{ pixels.}$$

Therefore, using (RESOLUTION÷2) as the number of pixels updated, the time required to update a 256 × 256 resolution 8-color display image is approximated by

$$\text{UPDATE} = \frac{65{,}536 \div 2}{24{,}600} = 1.332 \text{ seconds.}$$

Performance Summary

The iSBX 275 board is capable of drawing at a rate of 100,000 pixels per second in black-and-white mode. This assumes a bandwidth of 12.6 MHz. By changing the oscillator to allow for a 10 MHz bandwidth, the iSBX 275 board is capable of drawing at a rate of 40,000 pixels per second in color mode.

CONCLUSIONS

The iSBX 275 board provides an easy-to-use, configurable graphics capability for boards and systems which support the SBX bus. Color, resolution, and performance are design criteria which can be weighed against each other.

With the addition of the graphics routines described in this application note, the iSBX 275 board reduces graphical image generation to little more than a series of PL/M-86 calls. The two application examples verified that display images of varying complexity can be drawn using a handful of simple figure drawing commands. Circles, arcs, lines, rectangles, characters, character strings, and filled rectangular areas are each drawn by calling a single command. In addition, a single command is all that is required to change the drawing color, line style, or fill pattern, or to define a random 8 × 8 symbol.

Not only is the iSBX 275 board easy to use, but the general board setup and initialization sequence described in this application note will be required by only a handful of users. Most users will find that they can simply use the initializaiton parameters calculated in the two application examples, without going through the process of calculating the parameters for themselves.

APPENDIX A

PLANE DRAW$PANEL listing

```
                $eject

                /*****************************************************************
                                        Draw Panel

                *****************************************************************/
820     1       draw$panel: procedure ;
821     2           declare n byte;

                    /* clear memory and display */
822     2           call upcmd (0, 0, bw_plane);
823     2           call awcmd (replace,east,zeros, 4000h);

                    /* set pattern for full line style */
824     2           call agcmd (0ffffh, 0ffffh, 0ffffh, 0ffffh);

                    /* draw com control */
825     2           call umcmd (bw, set);
826     2           call ubcmd (088,112,65,35,east);
827     2           call ubcmd (095,119,51,11,east);
828     2           call strcmd (104,136,normal_char,east,one_x,bw,4,@com1$label);
829     2           call strcmd (120,120,normal_char,east,one_x,bw,1,@decimal);

830     2           do n = 0 to 4;
831     3               call strcmd (com$locn(n),120,normal_char,east,one_x,bw,
                                1,@digits(com$freq(n)));
832     3           end;

                    /* draw nav 1 control */
833     2           call agcmd (0ffffh,0ffffh,0ffffh,0ffffh);
834     2           call ubcmd (088,072,65,35,east);
835     2           call ubcmd (095,079,51,11,east);
836     2           call strcmd (104,096,normal_char,east,one_x,bw,4,@nav1$label);
837     2           call strcmd (120,080,normal_char,east,one_x,bw,1,@decimal);

838     2           do n = 0 to 4;
839     3               call strcmd (nav1$locn(n),80,normal_char,east,one_x,bw,
                                1,@digits(nav1$freq(n)));
840     3           end;

                    /* draw nav 2 control */
841     2           call agcmd (0ffffh,0ffffh,0ffffh,0ffffh);
842     2           call ubcmd (088,032,65,35,east);
843     2           call ubcmd (095,039,51,11,east);
844     2           call strcmd (104,056,normal_char,east,one_x,bw,4,@nav2$label);
845     2           call strcmd (120,040,normal_char,east,one_x,bw,1,@decimal);

846     2           do n = 0 to 4;
847     3               call strcmd (nav2$locn(n),40,normal_char,east,one_x,bw,
                                1,@digits(nav2$freq(n)));
848     3           end;

                    /* draw adf control */
849     2           call agcmd (0ffffh,0ffffh,0ffffh,0ffffh);
850     2           call ubcmd (168,112,65,35,east);
851     2           call ubcmd (183,119,35,11,east);
852     2           call strcmd (188,136,normal_char,east,one_x,bw,3,@adf$label);
853     2           do n = 0 to 3;
854     3               call strcmd (adf$locn(n),120,normal_char,east,one_x,bw,
                                1,@digits(adf$freq(n)));
855     3           end;
```

```
                /* draw transponder control */
856   2         call agcmd (0ffffh,0ffffh,0ffffh,0ffffh);
857   2         call ubcmd (168,072,65,35,east);
858   2         call ubcmd (183,079,35,11,east);
859   2         call strcmd (180,096,normal_char,east,one_x,bw,
                        5,@trans$label);

860   2         do n = 0 to 3;
861   3             call strcmd (tran$locn(n),80,normal_char,east,one_x,bw,
                            1,@digits(tran$freq(n)));
862   3         end;

                /* draw key board control */
863   2         call agcmd (0ffffh,0ffffh,0ffffh,0ffffh);
864   2         call ubcmd (168,032,65,35,east);
865   2         call ubcmd (175,039,51,11,east);
866   2         call strcmd (176,056,normal_char,east,one_x,bw,
                        6,@keybrd$label);
867   2         call strcmd (200,40,normal_char,east,one_x,bw,1,@decimal);

                /* draw marker beacon */
868   2         call agcmd (0ffffh,0ffffh,0ffffh,0ffffh);
869   2         call ubcmd (248,380,81,45,east);
870   2         call ubcmd (256,384,17,17,east);
871   2         call ubcmd (280,384,17,17,east);
872   2         call ubcmd (304,384,17,17,east);
873   2         call strcmd (260,408,normal_char,east,one_x,bw,1,@o$label);
874   2         call strcmd (284,408,normal_char,east,one_x,bw,1,@m$label);
875   2         call strcmd (308,408,normal_char,east,one_x,bw,1,@i$label);

                /* draw timer */
876   2         call agcmd (0ffffh,0ffffh,0ffffh,0ffffh);
877   2         call ubcmd (439,391,43,11,east);
878   2         call ubcmd (432,384,57,35,east);
879   2         call strcmd (440,408,normal_char,east,one_x,bw,
                        5,@timer$label);
880   2         call strcmd (456,392,normal_char,east,one_x,bw,1,@colon);

                /* draw circle for airspeed indicator */
881   2         call agcmd (0ffffh, 0ffffh, 0ffffh, 0ffffh);
882   2         call uccmd (073h, 144h, 34h);

883   2         call ulcmd (116,376,116,374);
884   2         call ulcmd (151,359,153,361);
885   2         call ulcmd (157,354,158,355);
886   2         call ulcmd (161,347,162,348);
887   2         call ulcmd (165,340,165,340);
888   2         call ulcmd (165,332,167,332);
889   2         call ulcmd (167,324,168,324);
890   2         call ulcmd (165,316,167,316);
891   2         call ulcmd (165,308,165,308);
892   2         call ulcmd (161,301,162,300);
893   2         call ulcmd (157,294,158,293);
894   2         call ulcmd (151,289,153,287);
895   2         call ulcmd (146,283,147,282);
896   2         call ulcmd (139,279,140,278);
897   2         call ulcmd (132,275,132,275);
898   2         call ulcmd (124,275,124,273);
899   2         call ulcmd (116,273,116,272);
900   2         call ulcmd (108,275,108,273);
901   2         call ulcmd (100,275,100,275);
902   2         call ulcmd ( 93,279, 92,278);
903   2         call ulcmd ( 86,283, 85,282);
904   2         call ulcmd ( 81,289, 79,287);
905   2         call ulcmd ( 75,294, 74,293);
906   2         call ulcmd ( 71,301, 70,300);
```

```
907   2          call ulcmd ( 67,308, 67,308);
908   2          call ulcmd ( 67,316, 65,316);
909   2          call ulcmd ( 65,324, 64,324);
910   2          call ulcmd ( 67,332, 65,332);

911   2          call strcmd (112,364,normal_char,east,one_x,bw,1,@zero);
912   2          call strcmd (136,348,normal_char,east,one_x,bw,2,@fifty);
913   2          call strcmd (148,320,normal_char,east,one_x,bw,
                            2,@seventyfive);
914   2          call strcmd (132,292,normal_char,east,one_x,bw,3,@hundred);
915   2          call strcmd (104,276,normal_char,east,one_x,bw,
                            3,@onetwentyfive);
916   2          call strcmd ( 76,292,normal_char,east,one_x,bw,3,@onefifty);
917   2          call strcmd ( 68,320,normal_char,east,one_x,bw,
                            3,@oneseventyfive);

                /* draw turn and bank indicator */
918   2          call agcmd (0ffffh, 0ffffh, 0ffffh, 0ffffh);
919   2          call uccmd (073h, 0d8h, 34h);
920   2          call ubcmd (05ah, 0b7h, 51, 18, east);
921   2          call ufcmd (06fh,0f8h,8,9,east);
922   2          call ulcmd (063h,0f4h,06ah,0f6h);
923   2          call ulcmd (06ah,0f6h,070h,0feh);
924   2          call ulcmd (070h,0feh,060h,0fch);
925   2          call ulcmd (060h,0fch,063h,0f4h);
926   2          call ulcmd (083h,0f4h,086h,0fch);
927   2          call ulcmd (086h,0fch,07fh,0feh);
928   2          call ulcmd (07fh,0feh,07dh,0f6h);
929   2          call ulcmd (07dh,0f6h,083h,0f4h);

930   2          do n = 0 to 7;
931   3              call ulcmd(tabx1(n),taby1(n),tabx2(n),taby2(n));
932   3          end;

933   2          call ufcmd (06fh,0b8h,8,9,east);

                /* draw artificial horizen indicator */
934   2          call uccmd (0e6h, 144h, 34h);

                /* draw directional gyro */
935   2          call uccmd (0e6h, 0d8h, 34h);
936   2          call ubcmd (0c7h, 0d4h, 63, 17, east);

937   2          call ulcmd (0e6h,0d8h,0e6h,0d6h);

                /* draw altimeter */
938   2          call uccmd (159h, 144h, 34h);
939   2          call strcmd (155h,170h,normal_char,east,one_x,bw,
                             length(zero), @zero);
940   2          call strcmd (171h,166h,normal_char,east,one_x,bw,
                             length(one), @one);
941   2          call strcmd (183h,150h,normal_char,east,one_x,bw,
                             length(two), @two);
942   2          call strcmd (183h,130h,normal_char,east,one_x,bw,
                             length(three), @three);
943   2          call strcmd (171h,118h,normal_char,east,one_x,bw,
                             length(four), @four);
944   2          call strcmd (155h,111h,normal_char,east,one_x,bw,
                             length(five), @five);
945   2          call strcmd (139h,118h,normal_char,east,one_x,bw,
                             length(six), @six);
946   2          call strcmd (12bh,130h,normal_char,east,one_x,bw,
                             length(seven), @seven);
947   2          call strcmd (12bh,150h,normal_char,east,one_x,bw,
                             length(eight), @eight);
948   2          call strcmd (139h,166h,normal_char,east,one_x,bw,
                             length(nine), @nine);
```

```
                /* draw rate of climb */
949     2       call agcmd (0ffffh, 0ffffh, 0ffffh, 0ffffh);
950     2       call uccmd (159h, 0d8h, 34h);

951     2       call strcmd (155h,0fch,normal_char,east,one_x,bw,
                             length(one), @one);
952     2       call strcmd (167h,0f4h,normal_char,east,one_x,bw,
                             length(one$point$five), @one$point$five);
953     2       call strcmd (17eh,0d4h,normal_char,east,one_x,bw,
                             length(two), @two);
954     2       call strcmd (167h,0b8h,normal_char,east,one_x,bw,
                             length(one$point$five), @one$point$five);
955     2       call strcmd (155h,0adh,normal_char,east,one_x,bw,
                             length(one), @one);
956     2       call strcmd (131h,0f4h,normal_char,east,one_x,bw,
                             length(half), @half);
957     2       call strcmd (131h,0b8h,normal_char,east,one_x,bw,
                             length(half), @half);
958     2       call strcmd (12ch,0d4h,normal_char,east,one_x,bw,
                             length(zero), @zero);

                /* draw tachometer */
959     2       call agcmd(0ffffh, 0ffffh, 0ffffh, 0ffffh);
960     2       call uccmd (159h, 06ch, 34h);
961     2       call ulcmd (308,71,309,72);
962     2       call ulcmd (303,77,304,78);
963     2       call ulcmd (298,84,300,85);
964     2       call ulcmd (295,92,297,92);
965     2       call ulcmd (293,100,295,100);
966     2       call ulcmd (293,108,295,108);
967     2       call ulcmd (293,116,295,116);
968     2       call ulcmd (295,124,297,123);
969     2       call ulcmd (298,132,300,131);
970     2       call ulcmd (303,139,305,137);
971     2       call ulcmd (308,145,310,143);
972     2       call ulcmd (314,150,316,148);
973     2       call ulcmd (321,154,322,153);
974     2       call ulcmd (329,157,330,156);
975     2       call ulcmd (337,159,337,157);
976     2       call ulcmd (345,160,345,158);
977     2       call ulcmd (353,159,353,157);
978     2       call ulcmd (361,157,360,156);
979     2       call ulcmd (369,154,368,153);
980     2       call ulcmd (376,150,374,148);
981     2       call ulcmd (382,145,380,143);
982     2       call ulcmd (387,139,385,137);
983     2       call ulcmd (391,132,390,131);
984     2       call ulcmd (394,124,393,123);
985     2       call ulcmd (396,116,394,116);
986     2       call ulcmd (397,108,395,108);
987     2       call ulcmd (396,100,394,100);
988     2       call ulcmd (394,92,393,93);
989     2       call ulcmd (391,84,390,85);
990     2       call ulcmd (387,77,385,79);
991     2       call ulcmd (382,71,380,73);
992     2       call strcmd (309,72,normal_char,east,one_x,bw,1,@zero);
993     2       call strcmd (297,104,normal_char,east,one_x,bw,1,@five);
994     2       call strcmd (309,136,normal_char,east,one_x,bw,2,@ten);
995     2       call strcmd (337,148,normal_char,east,one_x,bw,2,@fifteen);
996     2       call strcmd (365,136,normal_char,east,one_x,bw,2,@twenty);
997     2       call strcmd (377,104,normal_char,east,one_x,bw,2,@twentyfive);
998     2       call strcmd (365,72,normal_char,east,one_x,bw,2,@thirty);
999     2       call strcmd (333,80,normal_char,east,one_x,bw,3,@rpm$label);

                /* draw vor #1 indicator */
1000    2       call agcmd (0ffffh, 0ffffh, 0ffffh, 0ffffh);
```

```
1001   2          call uccmd (1cch, 144h, 34h);
1002   2          call ulcmd (456,327,456,321);
1003   2          call ulcmd (457,328,463,328);
1004   2          call ulcmd (464,327,464,321);
1005   2          call ulcmd (457,320,463,320);
1006   2          call strcmd (428,304,normal_char,east,one_x,bw,3,@obs$label);
1007   2          call strcmd (468,348,normal_char,east,one_x,bw,3,@dst$label);
1008   2          call strcmd (428,348,normal_char,east,one_x,bw,3,@nav$label);
1009   2          call strcmd (436,340,normal_char,east,one_x,bw,1,@one);

                   /* draw vor #2 indicator */
1010   2          call agcmd (0fffffh, 0fffffh, 0fffffh, 0fffffh);
1011   2          call uccmd (1cch, 0d8h, 34h);
1012   2          call ulcmd (456,219,456,213);
1013   2          call ulcmd (457,220,463,220);
1014   2          call ulcmd (464,219,464,213);
1015   2          call ulcmd (457,212,463,212);
1016   2          call strcmd (428,196,normal_char,east,one_x,bw,3,@obs$label);
1017   2          call strcmd (428,240,normal_char,east,one_x,bw,3,@nav$label);
1018   2          call strcmd (436,232,normal_char,east,one_x,bw,1,@two);

                   /* draw adf indicator */
1019   2          call agcmd (0fffffh,0fffffh,0fffffh,0fffffh);
1020   2          call uccmd (1cch, 06ch, 34h);
1021   2          call ulcmd (460,160,460,158);
1022   2          call ulcmd (469,159,468,157);
1023   2          call ulcmd (477,156,477,154);
1024   2          call ulcmd (485,153,484,151);
1025   2          call ulcmd (493,147,492,146);
1026   2          call ulcmd (499,141,498,140);
1027   2          call ulcmd (505,134,503,133);
1028   2          call ulcmd (508,125,506,125);
1029   2          call ulcmd (511,117,509,116);
1030   2          call ulcmd (512,108,510,108);
1031   2          call ulcmd (511, 98,509, 99);
1032   2          call ulcmd (508, 90,506, 90);
1033   2          call ulcmd (505, 82,503, 83);
1034   2          call ulcmd (499, 74,498, 75);
1035   2          call ulcmd (493, 68,492, 69);
1036   2          call ulcmd (486, 62,485, 64);
1037   2          call ulcmd (477, 59,477, 61);
1038   2          call ulcmd (469, 56,468, 58);
1039   2          call ulcmd (460, 56,460, 58);
1040   2          call ulcmd (450, 56,451, 58);
1041   2          call ulcmd (442, 59,442, 61);
1042   2          call ulcmd (434, 62,435, 64);
1043   2          call ulcmd (426, 68,427, 69);
1044   2          call ulcmd (420, 74,421, 75);
1045   2          call ulcmd (414, 81,416, 82);
1046   2          call ulcmd (411, 90,413, 90);
1047   2          call ulcmd (408, 98,410, 99);
1048   2          call ulcmd (408,108,410,108);
1049   2          call ulcmd (408,117,410,116);
1050   2          call ulcmd (411,125,413,125);
1051   2          call ulcmd (414,133,416,132);
1052   2          call ulcmd (420,141,421,140);
1053   2          call ulcmd (426,147,427,146);
1054   2          call ulcmd (433,153,434,151);
1055   2          call ulcmd (442,156,442,154);
1056   2          call ulcmd (450,159,451,157);
1057   2          call strcmd (456,148,normal_char,east,one_x,bw,1,@zero);
1058   2          call strcmd (478,142,normal_char,east,one_x,bw,1,@three);
1059   2          call strcmd (494,126,normal_char,east,one_x,bw,1,@six);
1060   2          call strcmd (500,104,normal_char,east,one_x,bw,1,@nine);
1061   2          call strcmd (488, 88,normal_char,east,one_x,bw,2,@twelve);
1062   2          call strcmd (476, 68,normal_char,east,one_x,bw,2,@fifteen);
```

```
1063   2            call strcmd (452,60,normal_char,east,one_x,bw,2,@eighteen);
1064   2            call strcmd (429,68,normal_char,east,one_x,bw,2,@twentyone);
1065   2            call strcmd (416,84,normal_char,east,one_x,bw,2,@twentyfour);
1066   2            call strcmd (412,104,normal_char,east,one_x,bw,
                                 2,@twentyseven);
1067   2            call strcmd (416,126,normal_char,east,one_x,bw,2,@thirty);
1068   2            call strcmd (429,142,normal_char,east,one_x,bw,
                                 2,@thirtythree);
1069   2            call strcmd (448,80,normal_char,east,one_x,bw,3,@adf$label);


1070   2            call agcmd (0ffffh, 0ffffh, 0ffffh, 0ffffh);

1071   2            call umcmd(bw,compliment);
1072   2       end draw$panel;
```

PLANE ALTIMETER listing

```
                $eject

                /*****************************************************************

                This is the altimeter display program. It is called using the
                actual altitude as a parameter.

                *****************************************************************/
615   1         altimeter: procedure (altitude) ;

616   2             declare altitude real;
617   2             declare (x21,y21,x22,y22,x23,y23) word;

618   2             declare x$org literally '159h';
619   2             declare y$org literally '144h';

                    /* save old values */
620   2             x21 = x$alt$10k;
621   2             x22 = x$alt$1k;
622   2             x23 = x$alt$100;
623   2             y21 = y$alt$10k;
624   2             y22 = y$alt$1k;
625   2             y23 = y$alt$100;

                    /* draw ten thousand foot pointer */
626   2             x$alt$10k = unsign(fix(16. * mqerCOS((90. - ((altitude/10000.)
                          * 36.)) * deg_to_rad))) + x$org;
627   2             y$alt$10k = unsign(fix(16. * mqerSIN((90. - ((altitude/10000.)
                          * 36.)) * deg_to_rad))) + y$org;
628   2             if (altitude > 10000.) then altitude = altitude - 10000.;

                    /* draw one thousand foot pointer */
630   2             x$alt$1k = unsign(fix(28. * mqerCOS((90. - ((altitude/1000.)
                         * 36.)) * deg_to_rad))) + x$org;
631   2             y$alt$1k = unsign(fix(28. * mqerSIN((90. - ((altitude/1000.)
                         * 36.)) * deg_to_rad))) + y$org;
632   2             do while (altitude > 1000.);
633   3                 altitude = altitude - 1000.;
634   3                 end;

                    /* draw one hundred foot pointer */
635   2             x$alt$100 = unsign(fix(40. * mqerCOS((90. - ((altitude/100.)
                          * 36.)) * deg_to_rad))) + x$org;
636   2             y$alt$100 = unsign(fix(40. * mqerSIN((90. - ((altitude/100.)
                          * 36.)) * deg_to_rad))) + y$org;

                    /* erase old hands */
637   2             call agcmd (0fffffh, 0fffffh, 0fffffh, 0fffffh);
638   2             call umcmd (bw, clear);
639   2             call ulcmd (x$org,y$org,x21,y21);
640   2             call ulcmd (x$org,y$org,x22,y22);
641   2             call ulcmd (x$org,y$org,x23,y23);

                    /* draw new hands */
642   2             call umcmd (bw, set);
643   2             call ulcmd (x$org,y$org,x$alt$10k,y$alt$10k);
644   2             call ulcmd (x$org,y$org,x$alt$1k,y$alt$1k);
645   2             call ulcmd (x$org,y$org,x$alt$100,y$alt$100);

                    /* return */
646   2             return;

647   2         end altimeter;
```

APPENDIX B

REACTOR listing

```
iRMX 86 PL/M-86 V2.3 COMPILATION OF MODULE REACTORMODULE
OBJECT MODULE PLACED IN /USER/SBX275/OBJ/REACTOR
COMPILER INVOKED BY:  :LANG:plm86 /USER/SBX275/SRC/REACTOR PRINT(/USER/SBX275/LS
T/REACTOR)
                      OBJECT(/USER/SBX275/OBJ/REACTOR) LARGE


   1          reactor$module: do;
              $nolist

  95    1       reactor: procedure reentrant public;

  96    2       declare   cycle word,
                          (i,j,k,l) byte,
                          (x1,y1,x2,y2) word;

  97    2       declare pattern (4) word data (4224h,
                                                 8a92h,
                                                 2429h,
                                                 484ah);

  98    2       declare line1 (*) byte data ('286 SYSTEMS'),
                        line2 (*) byte data ('in'),
                        line3 (*) byte data ('INDUSTRIAL CONTROL');

                  /********* DRAW OUTLINE *********/

                  /* set color and write mode */
  99    2         call umcmd(white,set);

                  /* draw reactor vessel outline */
 100    2         call ulcmd(23,152,23,158);
 101    2         call ulcmd(23,158,9,172);
 102    2         call ulcmd(9,172,9,203);
 103    2         call ulcmd(9,203,11,205);
 104    2         call ulcmd(11,205,44,205);
 105    2         call ulcmd(44,205,46,203);
 106    2         call ulcmd(46,203,46,172);
 107    2         call ulcmd(46,172,32,158);
 108    2         call ulcmd(32,158,32,152);

 109    2         call ulcmd(32,141,32,136);
 110    2         call ulcmd(33,135,50,135);
 111    2         call ulcmd(50,135,50,173);
 112    2         call ulcmd(50,173,52,175);
 113    2         call ulcmd(52,175,78,175);
 114    2         call ulcmd(78,175,80,173);
 115    2         call ulcmd(80,173,80,165);
 116    2         call ulcmd(80,165,151,165);
 117    2         call ulcmd(152,164,153,164);
 118    2         call ulcmd(154,163,155,163);
 119    2         call ulcmd(155,163,155,162);
 120    2         call ulcmd(156,161,156,158);
 121    2         call ulcmd(155,157,155,156);
 122    2         call ulcmd(155,156,147,148);

 123    2         call ulcmd(144,150,156,138);
 124    2         call ulcmd(156,137,157,136);
 125    2         call ulcmd(157,135,156,134);
 126    2         call ulcmd(156,133,147,124);
```

```
127   2          call ulcmd(144,126,155,115);
128   2          call ulcmd(155,115,155,114);
129   2          call ulcmd(156,113,156,110);
130   2          call ulcmd(155,109,155,108);
131   2          call ulcmd(155,108,154,108);
132   2          call ulcmd(153,107,152,107);
133   2          call ulcmd(151,106,125,106);

134   2          call ulcmd(23,141,23,128);
135   2          call ulcmd(23,128,25,126);
136   2          call ulcmd(25,126,50,126);
137   2          call ulcmd(50,126,50,101);
138   2          call ulcmd(50,101,52,99);
139   2          call ulcmd(52,99,78,99);
140   2          call ulcmd(78,99,80,101);
141   2          call ulcmd(80,101,80,106);
142   2          call ulcmd(80,106,108,106);

143   2          call ulcmd(108,115,80,115);
144   2          call ulcmd(80,115,80,156);
145   2          call ulcmd(80,156,146,156);

146   2          call ulcmd(140,155,135,150);
147   2          call ulcmd(135,149,134,148);
148   2          call ulcmd(134,147,135,146);
149   2          call ulcmd(135,145,147,133);

150   2          call ulcmd(144,135,135,126);
151   2          call ulcmd(135,125,134,124);
152   2          call ulcmd(134,123,135,122);
153   2          call ulcmd(135,121,144,112);

154   2          call ulcmd(141,115,125,115);

                /* draw boxes on reactor vessel sides */
155   2          call ufcmd(47,146,3,8,east);
156   2          call ufcmd(81,146,3,8,east);

                /* draw steam generator outline */
157   2          call ulcmd(100,152,100,155);
158   2          call ulcmd(92,166,86,172);
159   2          call ulcmd(86,172,86,203);
160   2          call ulcmd(86,203,88,205);
161   2          call ulcmd(88,205,121,205);
162   2          call ulcmd(121,205,123,203);
163   2          call ulcmd(123,203,123,172);
164   2          call ulcmd(123,172,117,166);
165   2          call ulcmd(109,155,109,152);

166   2          call ulcmd(109,141,109,136);
167   2          call ulcmd(110,135,129,135);
168   2          call ulcmd(129,135,129,155);
169   2          call ulcmd(129,166,129,188);
170   2          call ulcmd(129,188,131,190);
171   2          call ulcmd(131,190,141,190);
172   2          call ulcmd(141,190,141,203);
173   2          call ulcmd(141,203,143,205);
174   2          call ulcmd(143,205,195,205);
175   2          call ulcmd(195,205,197,203);
176   2          call ulcmd(197,203,197,194);

177   2          call ulcmd(100,141,100,128);
178   2          call ulcmd(100,128,102,126);
179   2          call ulcmd(102,126,129,126);
180   2          call ulcmd(129,126,129,116);
181   2          call ulcmd(129,105,129,101);
```

```
182   2           call ulcmd(129,101,131,99);
183   2           call ulcmd(131,99,141,99);
184   2           call ulcmd(141,99,141,82);
185   2           call ulcmd(141,82,143,80);
186   2           call ulcmd(143,80,172,80);

187   2           call ulcmd(172,89,151,89);
188   2           call ulcmd(150,90,150,99);
189   2           call ulcmd(150,99,160,99);
190   2           call ulcmd(160,99,162,101);
191   2           call ulcmd(162,101,162,188);
192   2           call ulcmd(162,188,160,190);
193   2           call ulcmd(160,190,150,190);
194   2           call ulcmd(150,190,150,195);
195   2           call ulcmd(151,196,187,196);
196   2           call ulcmd(188,195,188,191);

197   2           call ulcmd(203,167,203,159);
198   2           call ulcmd(203,159,193,159);
199   2           call ulcmd(193,159,191,157);
200   2           call ulcmd(191,157,191,101);
201   2           call ulcmd(191,101,193,99);
202   2           call ulcmd(193,99,203,99);
203   2           call ulcmd(203,99,203,90);
204   2           call ulcmd(202,89,189,89);

205   2           call ulcmd(189,80,210,80);
206   2           call ulcmd(210,80,212,82);
207   2           call ulcmd(212,82,212,99);
208   2           call ulcmd(212,99,222,99);
209   2           call ulcmd(222,99,224,101);
210   2           call ulcmd(224,101,224,115);
211   2           call ulcmd(224,126,224,141);
212   2           call ulcmd(224,152,224,157);
213   2           call ulcmd(224,157,222,159);
214   2           call ulcmd(222,159,212,159);
215   2           call ulcmd(212,159,212,164);

                  /* draw condenser pipe */
216   2           call ulcmd(287,151,202,151);
217   2           call ulcmd(201,150,200,150);
218   2           call ulcmd(199,149,198,149);
219   2           call ulcmd(198,149,198,148);
220   2           call ulcmd(197,147,197,144);
221   2           call ulcmd(198,143,198,142);
222   2           call ulcmd(198,142,206,134);

223   2           call ulcmd(209,136,198,125);
224   2           call ulcmd(198,125,198,124);
225   2           call ulcmd(197,123,197,120);
226   2           call ulcmd(198,119,198,118);
227   2           call ulcmd(198,118,199,118);
228   2           call ulcmd(200,117,201,117);
229   2           call ulcmd(202,116,287,116);

230   2           call ulcmd(287,125,212,125);

231   2           call ulcmd(209,122,218,131);
232   2           call ulcmd(218,132,219,133);
233   2           call ulcmd(219,134,218,135);
234   2           call ulcmd(218,136,213,141);

235   2           call ulcmd(207,142,287,142);

                  /********* DRAW OBJECTS *********/
```

```
                /* draw turbine-to-generator shaft */
236   2         call ufcmd(220,179,10,6,east);

                /* change color */
237   2         call umcmd(red,set);

                /* draw turbine */
238   2         x1 = 181;
239   2         y1 = 175;
240   2         y2 = 14;

241   2         do while x1 < 220;
242   3           call ufcmd(x1,y1,3,y2,east);
243   3           x1 = x1 + 3;
244   3           y1 = y1 - 1;
245   3           y2 = y2 + 2;
246   3         end;

                /* change color */
247   2         call umcmd(green,set);

                /* draw generator */
248   2         call ufcmd(230,166,48,32,east);
249   2         call ufcmd(231,165,46,34,east);
250   2         call ufcmd(232,164,44,36,east);

                /* draw valve and pump blocks */
251   2         call ufcmd(22,142,12,10,east);
252   2         call ufcmd(99,142,12,10,east);
253   2         call ufcmd(109,105,16,12,east);
254   2         call ufcmd(173,79,16,12,east);

                /* change color */
255   2         call umcmd(yellow,set);
                /* draw emergency fluid valves */
256   2         call ulcmd(24,150,31,143);
257   2         call ulcmd(101,150,108,143);

                /* draw reactor pump gear */
258   2         call ufcmd(115,107,4,8,east);
259   2         call ufcmd(113,109,8,4,east);
260   2         call ufcmd(116,106,2,10,east);
261   2         call ufcmd(112,110,10,2,east);

                /* draw steam generator pump gear */
262   2         call ufcmd(179,81,4,8,east);
263   2         call ufcmd(177,83,8,4,east);
264   2         call ufcmd(180,80,2,10,east);
265   2         call ufcmd(176,84,10,2,east);

                /* change color */
266   2         call umcmd(magenta,set);

                /* draw stationary elements inside reactor */
267   2         call ufcmd(53,124,5,14,east);
268   2         call ufcmd(63,124,5,14,east);
269   2         call ufcmd(73,124,5,14,east);

                /* change color */
270   2         call umcmd(cyan,set);

                /* draw moveable elements inside reactor */
271   2         call ufcmd(59,131,3,14,east);
272   2         call ufcmd(69,131,3,14,east);
273   2         call ulcmd(60,145,60,198);
274   2         call ulcmd(70,145,70,198);
```

```
                /********* FILL WITH FLUID *********/

                /* change color */
275   2         call umcmd(blue,set);

                /* fill reactor emergency fluid container */
276   2         call ufcmd(10,171,36,30,east);
277   2         call ufcmd(27,155,16,16,northeast);
278   2         call ufcmd(28,155,16,16,northeast);
279   2         call ufcmd(24,152,8,7,east);
280   2         call ufcmd(24,127,9,15,east);
281   2         call ufcmd(33,127,18,8,east);

                /* fill reactor */
282   2         call ufcmd(80,107,29,8,east);
283   2         call ufcmd(51,100,29,75,east);
284   2         call ufcmd(80,157,74,8,east);
285   2         call ufcmd(154,157,2,6,east);
286   2         call ufcmd(140,142,15,7,northeast);
287   2         call ufcmd(139,142,15,6,northeast);
288   2         call ufcmd(152,130,6,12,northeast);
289   2         call ufcmd(151,130,7,12,northeast);
290   2         call ufcmd(140,118,12,7,northeast);
291   2         call ufcmd(139,118,12,6,northeast);
292   2         call ufcmd(148,109,7,9,northeast);
293   2         call ufcmd(148,110,6,8,northeast);
294   2         call ufcmd(125,107,29,8,east);
295   2         call ufcmd(154,109,2,6,east);

                /* fill condenser pipe */
296   2         call ufcmd(200,143,88,8,east);
297   2         call ufcmd(198,143,2,6,east);
298   2         call ufcmd(214,128,6,15,northeast);
299   2         call ufcmd(213,128,7,15,northeast);
300   2         call ufcmd(205,119,9,7,northeast);
301   2         call ufcmd(205,120,8,6,northeast);
302   2         call ufcmd(200,117,88,8,east);
303   2         call ufcmd(198,119,2,6,east);

                /* fill steam generator emergency fluid container */
304   2         call ufcmd(87,171,36,30,east);
305   2         call ufcmd(89,169,32,2,east);
306   2         call ufcmd(91,167,28,2,east);
307   2         call ufcmd(93,166,24,1,east);
308   2         call ufcmd(101,152,8,4,east);
309   2         call ufcmd(101,127,9,15,east);
310   2         call ufcmd(110,127,20,8,east);

                /* fill steam generator liquid */
311   2         call ufcmd(192,100,32,12,east);
312   2         call ufcmd(204,81,8,19,east);
313   2         call ufcmd(189,81,15,9,east);
314   2         call ufcmd(142,81,31,8,east);
315   2         call ufcmd(142,89,9,11,east);
316   2         call ufcmd(130,100,32,76,east);

                /********* FILL WITH STEAM *********/

317   2     aa: y1 = 177;
318   2         k = 0;

319   2         do while y1 < 191;
320   3           call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
321   3           call ulcmd(130,y1,161,y1);
322   3           y1 = y1 + 2;
323   3           k = k + 1;
324   3         end;
```

```
325   2       bb: y1 = 191;

326   2           do while y1 < 197;
327   3             call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
328   3             call ulcmd(142,y1,149,y1);
329   3             y1 = y1 + 2;
330   3             k = k + 1;
331   3           end;

332   2       cc: y1 = 197;
333   2           x2 = 149;

334   2           do while y1 < 205;
335   3             call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
336   3             call ulcmd(142,y1,x2,y1);
337   3             y1 = y1 + 2;
338   3             x2 = x2 - 2;
339   3             k = k + 1;
340   3           end;

341   2       dd: x1 = 143;
342   2           y2 = 204;

343   2           do while x1 < 151;
344   3             call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
345   3             call ulcmd(x1,204,x1,y2);
346   3             x1 = x1 + 2;
347   3             y2 = y2 - 2;
348   3             k = k + 1;
349   3           end;

350   2       ee: x1 = 151;

351   2           do while x1 < 189;
352   3             call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
353   3             call ulcmd(x1,204,x1,197);
354   3             x1 = x1 + 2;
355   3             k = k + 1;
356   3           end;

357   2       ff: x1 = 189;
358   2           y2 = 197;

359   2           do while x1 < 197;
360   3             call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
361   3             call ulcmd(x1,204,x1,y2);
362   3             x1 = x1 + 2;
363   3             y2 = y2 + 2;
364   3             k = k + 1;
365   3           end;

366   2       gg: y1 = 203;
367   2           x2 = 196;

368   2           do while y1 > 195;
369   3             call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
370   3             call ulcmd(196,y1,x2,y1);
371   3             y1 = y1 - 2;
372   3             x2 = x2 - 2;
373   3             k = k + 1;
374   3           end;

375   2       hh: y1 = 195;

376   2           do while y1 > 191;
377   3             call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
```

```
378   3             call ulcmd(196,y1,189,y1);
379   3             y1 = y1 - 2;
380   3             k = k + 1;
381   3           end;
382   2       ii: y1 = 165;

383   2           do while y1 > 157;
384   3             call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
385   3             call ulcmd(204,y1,211,y1);
386   3             y1 = y1 - 2;
387   3             k = k + 1;
388   3           end;

389   2       jj: y1 = 157;

390   2           do while y1 > 151;
391   3             call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
392   3             call ulcmd(192,y1,223,y1);
393   3             y1 = y1 - 2;
394   3             k = k + 1;
395   3           end;

396   2       kk: call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
397   2           call ulcmd(192,151,201,151);
398   2           k = k + 1;

399   2       ll: y1 = 149;
400   2           x2 = 197;

401   2           do while y1 > 145;
402   3             call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
403   3             call ulcmd(192,y1,x2,y1);
404   3             y1 = y1 - 2;
405   3             x2 = x2 - 1;
406   3             k = k + 1;
407   3           end;

408   2       mm: y1 = 145;
409   2           x2 = 196;

410   2           do while y1 > 141;
411   3             call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
412   3             call ulcmd(192,y1,x2,y1);
413   3             y1 = y1 - 2;
414   3             x2 = x2 + 1;
415   3             k = k + 1;
416   3           end;

417   2       nn: y1 = 141;
418   2           x2 = 198;
419   2           x1 = 214;

420   2           do while y1 > 133;
421   3             call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
422   3             call ulcmd(192,y1,x2,y1);
423   3             call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
424   3             call ulcmd(x1,y1,223,y1);
425   3             y1 = y1 - 2;
426   3             x2 = x2 + 2;
427   3             x1 = x1 + 2;
428   3             k = k + 1;
429   3           end;

430   2       oo: y1 = 133;
431   2           x2 = 205;
432   2           x1 = 221;
```

```
433   2           do while y1 > 125;
434   3             call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
435   3             call ulcmd(192,y1,x2,y1);
436   3             call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
437   3             call ulcmd(x1,y1,223,y1);
438   3             y1 = y1 - 2;
439   3             x2 = x2 - 2;
440   3             x1 = x1 - 2;
441   3             k = k + 1;
442   3           end;

443   2       pp: y1 = 125;
444   2           x2 = 197;

445   2           do while y1 > 121;
446   3             call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
447   3             call ulcmd(192,y1,x2,y1);
448   3             y1 = y1 - 2;
449   3             x2 = x2 - 1;
450   3             k = k + 1;
451   3           end;

452   2       qq: y1 = 121;
453   2           x2 = 196;

454   2           do while y1 > 115;
455   3             call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
456   3             call ulcmd(192,y1,x2,y1);
457   3             y1 = y1 - 2;
458   3             x2 = x2 + 1;
459   3             k = k + 1;
460   3           end;

461   2       rr: y1 = 115;

462   2           do while y1 > 111;
463   3             call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
464   3             call ulcmd(192,y1,223,y1);
465   3             y1 = y1 - 2;
466   3             k = k + 1;
467   3           end;

                  /********* OUTPUT LABEL *********/

468   2           call strcmd(56,50,normal_char,east,two_x,red,
                              length(line1),@line1);
469   2           call strcmd(128,30,normal_char,east,two_x,blue,
                              length(line2),@line2);
470   2           call strcmd(0,10,normal_char,east,two_x,green,
                              length(line3),@line3);
                  /********* CAUSE MOTION *********/

                  /* initialize variable */
471   2           i = 0;

                  /* change write mode */
472   2           call umcmd(blue,complement);

473   2           do cycle = 0 to 2;

                    /* change line drawing pattern */
474   3             call agcmd(5555h,0ffffh,0ffffh,0ffffh);

475   3             i = ( (cycle and 3) / 2);
```

```
                    /* cause motion in reactor fluid */
476   3       aa1:  x1 = 108 - i;

477   3             do while x1 > 79;
478   4               call ulcmd(x1,107,x1,114);
479   4               x1 = x1 - 2;
480   4             end;

481   3       bb1:  y1 = 100 + i;

482   3             do while y1 < 175;
483   4               call ulcmd(51,y1,79,y1);
484   4               y1 = y1 + 2;
485   4             end;

486   3       cc1:  x1 = 80 + i;

487   3             do while x1 < 154;
488   4               call ulcmd(x1,157,x1,164);
489   4               x1 = x1 + 2;
490   4             end;

491   3       dd1:  x1 = 154 + i;

492   3             call ulcmd(x1,157,x1,162);

493   3       ee1:  x1 = 153 - (2 * i);
494   3             y1 = 155 - i;
495   3             x2 = 153 - (2 * i);
496   3             y2 = 155 - i;

497   3             do while y2 > 148;
498   4               call ulcmd(x1,y1,x2,y2);
499   4               x1 = x1 - 2;
500   4               x2 = x2 - 1;
501   4               y2 = y2 - 1;
502   4             end;

503   3       ff1:  x1 = 139;
504   3             y1 = 153 - i;
505   3             x2 = 134;
506   3             y2 = 148 - i;
507   3             j = i;

508   3             do while y2 > 129;
509   4               call ulcmd(x1,y1,x2,y2);
510   4               x2 = x2 + 1;
511   4               y2 = y2 - 1;
512   4               if (j and 1) = 0 then
513   4                 y1 = y1 - 2;
514   4               else
                        x1 = x1 + 2;
515   4               j = j + 1;
516   4             end;

517   3       gg1:  x1 = 145;
518   3             y1 = 134 - i;
519   3             x2 = 150;
520   3             y2 = 129 - i;
521   3             j = i;

522   3             do while y2 > 124;
523   4               call ulcmd(x1,y1,x2,y2);
524   4               x2 = x2 - 1;
525   4               y2 = y2 - 1;
526   4               if (j and 1) = 0 then
```

```
527   4                 y1 = y1 - 2;
528   4               else
                        x1 = x1 - 2;
529   4               j = j + 1;
530   4             end;

531   3     hh1:    x1 = 139;
532   3             y1 = 129 - i;
533   3             x2 = 134;
534   3             y2 = 124 - i;
535   3             j = i;

536   3             do while y2 > 106;
537   4               call ulcmd(x1,y1,x2,y2);
538   4               x2 = x2 + 1;
539   4               y2 = y2 - 1;
540   4               if (j and 1) = 0 then
541   4                 y1 = y1 - 2;
542   4               else
                        x1 = x1 + 2;
543   4               j = j + 1;
544   4             end;

545   3     ii1:    x1 = 148 - i;
546   3             y2 = 107 + i;

547   3             do while x1 > 140;
548   4               call ulcmd(x1,107,x1,y2);
549   4               x1 = x1 - 2;
550   4               y2 = y2 + 2;
551   4             end;
552   3     jj1:    x1 = 140 - i;

553   3             do while x1 > 124;
554   4               call ulcmd(x1,107,x1,114);
555   4               x1 = x1 - 2;
556   4             end;

                    /* cause motion in condenser pipe fluid */
557   3     aa2:    x1 = 287 - i;

558   3             do while x1 > 199;
559   4               call ulcmd(x1,143,x1,150);
560   4               x1 = x1 - 2;
561   4             end;

562   3     bb2:    x1 = 199 - i;

563   3             call ulcmd(x1,143,x1,148);

564   3     cc2:    x1 = 200 + (2 * i);
565   3             y1 = 141 - i;
566   3             x2 = 200 + (2 * i);
567   3             y2 = 141 - i;

568   3             do while y2 > 134;
569   4               call ulcmd(x1,y1,x2,y2);
570   4               x1 = x1 + 2;
571   4               x2 = x2 + 1;
572   4               y2 = y2 - 1;
573   4             end;

574   3     dd2:    x1 = 214;
575   3             y1 = 139 - i;
576   3             x2 = 219;
577   3             y2 = 134 - i;
```

```
578   3             j = i;

579   3             do while y2 > 116;
580   4               call ulcmd(x1,y1,x2,y2);
581   4               x2 = x2 - 1;
582   4               y2 = y2 - 1;
583   4               if (j and 1) = 0 then
584   4                 y1 = y1 - 2;
585   4               else
                        x1 = x1 - 2;
586   4               j = j + 1;
587   4             end;

588   3       ee2:  x1 = 205 + i;
589   3             y2 = 117 + i;

590   3             do while x1 < 213;
591   4               call ulcmd(x1,117,x1,y2);
592   4               x1 = x1 + 2;
593   4               y2 = y2 + 2;
594   4             end;
595   3       ff2:  x1 = 213 + i;

596   3             do while x1 < 288;
597   4               call ulcmd(x1,117,x1,124);
598   4               x1 = x1 + 2;
599   4             end;

                    /* cause motion in steam generator fluid */
600   3       aa3:  y1 = 111 - i;

601   3             do while y1 > 99;
602   4               call ulcmd(223,y1,192,y1);
603   4               y1 = y1 - 2;
604   4             end;

605   3       bb3:  y1 = 99 - i;

606   3             do while y1 > 89;
607   4               call ulcmd(211,y1,204,y1);
608   4               y1 = y1 - 2;
609   4             end;

610   3       cc3:  y1 = 89 - i;
611   3             x2 = 204 + i;

612   3             do while y1 > 81;
613   4               call ulcmd(211,y1,x2,y1);
614   4               y1 = y1 - 2;
615   4               x2 = x2 + 2;
616   4             end;

617   3       dd3:  x1 = 210 - i;
618   3             y2 = 81 + i;

619   3             do while x1 > 202;
620   4               call ulcmd(x1,81,x1,y2);
621   4               x1 = x1 - 2;
622   4               y2 = y2 + 2;
623   4             end;

624   3       ee3:  x1 = 202 - i;

625   3             do while x1 > 188;
626   4               call ulcmd(x1,81,x1,88);
```

```
627   4               x1 = x1 - 2;
628   4             end;

629   3       ff3:  x1 = 172 - i;

630   3             do while x1 > 150;
631   4               call ulcmd(x1,81,x1,88);
632   4               x1 = x1 - 2;
633   4             end;

634   3       gg3:  x1 = 150 - i;
635   3             y2 = 88 - i;
636   3             do while x1 > 142;
637   4               call ulcmd(x1,81,x1,y2);
638   4               x1 = x1 - 2;
639   4               y2 = y2 - 2;
640   4             end;

641   3       hh3:  y1 = 82 + i;
642   3             x2 = 142 + i;

643   3             do while y1 < 90;
644   4               call ulcmd(142,y1,x2,y1);
645   4               y1 = y1 + 2;
646   4               x2 = x2 + 2;
647   4             end;

648   3       ii3:  y1 = 90 + i;

649   3             do while y1 < 100;
650   4               call ulcmd(142,y1,149,y1);
651   4               y1 = y1 + 2;
652   4             end;

653   3       jj3:  y1 = 100 + i;

654   3             do while y1 < 106;
655   4               call ulcmd(130,y1,161,y1);
656   4               y1 = y1 + 2;
657   4             end;

658   3       kk3:  x1 = 154;
659   3             y1 = 106 + i;

660   3             do while y1 < 112;
661   4               call ulcmd(x1,y1,161,y1);
662   4               x1 = x1 + 2;
663   4               y1 = y1 + 2;
664   4             end;

665   3       ll3:  x1 = 158;
666   3             y1 = 112 + i;

667   3             do while y1 < 116;
668   4               call ulcmd(x1,y1,161,y1);
669   4               x1 = x1 - 2;
670   4               y1 = y1 + 2;
671   4             end;

672   3       mm3:  x1 = 154;
673   3             y1 = 116 + i;
674   3             x2 = 138;

675   3             do while y1 < 124;
676   4               call ulcmd(130,y1,x2,y1);
677   4               call ulcmd(x1,y1,161,y1);
```

```
678   4              x1 = x1 - 2;
679   4              y1 = y1 + 2;
680   4              x2 = x2 - 2;

681   4            end;

682   3     nn3:   x1 = 148;
683   3            y1 = 124 + i;
684   3            x2 = 132;

685   3            do while y1 < 136;
686   4              call ulcmd(130,y1,x2,y1);
687   4              call ulcmd(x1,y1,161,y1);
688   4              x1 = x1 + 2;
689   4              y1 = y1 + 2;
690   4              x2 = x2 + 2;
691   4            end;

692   3     oo3:   x1 = 158;
693   3            y1 = 136 + i;
694   3            x2 = 142;

695   3            do while y1 < 148;
696   4              call ulcmd(130,y1,x2,y1);
697   4              call ulcmd(x1,y1,161,y1);
698   4              x1 = x1 - 2;
699   4              y1 = y1 + 2;
700   4              x2 = x2 - 2;
701   4            end;

702   3     pp3:   x1 = 148;
703   3            y1 = 148 + i;
704   3            x2 = 132;

705   3            do while y1 < 156;
706   4              call ulcmd(130,y1,x2,y1);
707   4              call ulcmd(x1,y1,161,y1);
708   4              x1 = x1 + 2;
709   4              y1 = y1 + 2;
710   4              x2 = x2 + 2;
711   4            end;

712   3     qq3:   x1 = 156;
713   3            y1 = 156 + i;

714   3            do while y1 < 160;
715   4              call ulcmd(x1,y1,161,y1);
716   4              x1 = x1 + 2;
717   4              y1 = y1 + 2;
718   4            end;

719   3     rr3:   x1 = 158;
720   3            y1 = 160 + i;

721   3            do while y1 < 166;
722   4              call ulcmd(x1,y1,161,y1);
723   4              x1 = x1 - 2;
724   4              y1 = y1 + 2;
725   4            end;

726   3     ss3:   y1 = 166 + i;
727   3            do while y1 < 176;
728   4              call ulcmd(130,y1,161,y1);
729   4              y1 = y1 + 2;
730   4            end;
```

```
                    /* cause motion in steam */
731   3       as3:  y1 = 177;
732   3             k = 1;
733   3             l = l - 1;

734   3             do while y1 < 191;
735   4               call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
736   4               call ulcmd(130,y1,161,y1);
737   4               call agcmd( pattern((k - 1) and 3),0ffffh,0ffffh,0ffffh );
738   4               call ulcmd(130,y1,161,y1);
739   4               y1 = y1 + 2;
740   4               k = k + 1;
741   4             end;

742   3       bs3:  y1 = 191;

743   3             do while y1 < 197;
744   4               call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
745   4               call ulcmd(142,y1,149,y1);
746   4               call agcmd( pattern((k - 1) and 3),0ffffh,0ffffh,0ffffh );
747   4               call ulcmd(142,y1,149,y1);
748   4               y1 = y1 + 2;
749   4               k = k + 1;
750   4             end;

751   3       cs3:  y1 = 197;
752   3             x2 = 149;

753   3             do while y1 < 205;
754   4               call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
755   4               call ulcmd(142,y1,x2,y1);
756   4               call agcmd( pattern((k - 1) and 3),0ffffh,0ffffh,0ffffh );
757   4               call ulcmd(142,y1,x2,y1);
758   4               y1 = y1 + 2;
759   4               x2 = x2 - 2;
760   4               k = k + 1;
761   4             end;

762   3       ds3:  x1 = 143;
763   3             y2 = 204;

764   3             do while x1 < 151;
765   4               call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
766   4               call ulcmd(x1,204,x1,y2);
767   4               call agcmd( pattern((k - 1) and 3),0ffffh,0ffffh,0ffffh );
768   4               call ulcmd(x1,204,x1,y2);
769   4               x1 = x1 + 2;
770   4               y2 = y2 - 2;
771   4               k = k + 1;
772   4             end;
773   3       es3:  x1 = 151;

774   3             do while x1 < 189;
775   4               call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
776   4               call ulcmd(x1,204,x1,197);
777   4               call agcmd( pattern((k - 1) and 3),0ffffh,0ffffh,0ffffh );
778   4               call ulcmd(x1,204,x1,197);
779   4               x1 = x1 + 2;
780   4               k = k + 1;
781   4             end;

782   3       fs3:  x1 = 189;
783   3             y2 = 197;

784   3             do while x1 < 197;
785   4               call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
```

```
786   4              call ulcmd(x1,204,x1,y2);
787   4              call agcmd( pattern((k - 1) and 3),0ffffh,0ffffh,0ffffh );
788   4              call ulcmd(x1,204,x1,y2);
789   4              x1 = x1 + 2;
790   4              y2 = y2 + 2;
791   4              k = k + 1;
792   4            end;

793   3      gs3:  y1 = 203;
794   3            x2 = 196;

795   3            do while y1 > 195;
796   4              call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
797   4              call ulcmd(196,y1,x2,y1);
798   4              call agcmd( pattern((k - 1) and 3),0ffffh,0ffffh,0ffffh );
799   4              call ulcmd(196,y1,x2,y1);
800   4              y1 = y1 - 2;
801   4              x2 = x2 - 2;
802   4              k = k + 1;
803   4            end;

804   3      hs3:  y1 = 195;

805   3            do while y1 > 191;
806   4              call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
807   4              call ulcmd(196,y1,189,y1);
808   4              call agcmd( pattern((k - 1) and 3),0ffffh,0ffffh,0ffffh );
809   4              call ulcmd(196,y1,189,y1);
810   4              y1 = y1 - 2;
811   4              k = k + 1;
812   4            end;

813   3      is3:  y1 = 165;

814   3            do while y1 > 157;
815   4              call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
816   4              call ulcmd(204,y1,211,y1);
817   4              call agcmd( pattern((k - 1) and 3),0ffffh,0ffffh,0ffffh );
818   4              call ulcmd(204,y1,211,y1);
819   4              y1 = y1 - 2;
820   4              k = k + 1;
821   4            end;

822   3      js3:  y1 = 157;

823   3            do while y1 > 151;
824   4              call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
825   4              call ulcmd(192,y1,223,y1);
826   4              call agcmd( pattern((k - 1) and 3),0ffffh,0ffffh,0ffffh );
827   4              call ulcmd(192,y1,223,y1);
828   4              y1 = y1 - 2;
829   4              k = k + 1;
830   4            end;

831   3      ks3:  call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
832   3            call ulcmd(192,151,201,151);
833   3            call agcmd( pattern((k - 1) and 3),0ffffh,0ffffh,0ffffh );
834   3            call ulcmd(192,151,201,151);
835   3            k = k + 1;

836   3      ls3:  y1 = 149;
837   3            x2 = 197;

838   3            do while y1 > 145;
839   4              call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
840   4              call ulcmd(192,y1,x2,y1);
```

```
841   4                call agcmd( pattern((k - 1) and 3),0ffffh,0ffffh,0ffffh );
842   4                call ulcmd(192,y1,x2,y1);
843   4                y1 = y1 - 2;
844   4                x2 = x2 - 1;
845   4                k = k + 1;
846   4              end;

847   3      ms3:    y1 = 145;
848   3              x2 = 196;

849   3              do while y1 > 141;
850   4                call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
851   4                call ulcmd(192,y1,x2,y1);
852   4                call agcmd( pattern((k - 1) and 3),0ffffh,0ffffh,0ffffh );
853   4                call ulcmd(192,y1,x2,y1);
854   4                y1 = y1 - 2;
855   4                x2 = x2 + 1;
856   4                k = k + 1;
857   4              end;

858   3      ns3:    y1 = 141;
859   3              x2 = 198;
860   3              x1 = 214;

861   3              do while y1 > 133;
862   4                call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
863   4                call ulcmd(192,y1,x2,y1);
864   4                call agcmd( pattern((k - 1) and 3),0ffffh,0ffffh,0ffffh );
865   4                call ulcmd(192,y1,x2,y1);
866   4                call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
867   4                call ulcmd(x1,y1,223,y1);
868   4                call agcmd( pattern((k - 1) and 3),0ffffh,0ffffh,0ffffh );
869   4                call ulcmd(x1,y1,223,y1);
870   4                y1 = y1 - 2;
871   4                x2 = x2 + 2;
872   4                x1 = x1 + 2;
873   4                k = k + 1;
874   4              end;

875   3      os3:    y1 = 133;
876   3              x2 = 205;
877   3              x1 = 221;

878   3              do while y1 > 125;
879   4                call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
880   4                call ulcmd(192,y1,x2,y1);
881   4                call agcmd( pattern((k - 1) and 3),0ffffh,0ffffh,0ffffh );
882   4                call ulcmd(192,y1,x2,y1);
883   4                call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
884   4                call ulcmd(x1,y1,223,y1);
885   4                call agcmd( pattern((k - 1) and 3),0ffffh,0ffffh,0ffffh );
886   4                call ulcmd(x1,y1,223,y1);
887   4                y1 = y1 - 2;
888   4                x2 = x2 - 2;
889   4                x1 = x1 - 2;
890   4                k = k + 1;
891   4              end;

892   3      ps3:    y1 = 125;
893   3              x2 = 197;

894   3              do while y1 > 121;
895   4                call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
896   4                call ulcmd(192,y1,x2,y1);
897   4                call agcmd( pattern((k - 1) and 3),0ffffh,0ffffh,0ffffh );
898   4                call ulcmd(192,y1,x2,y1);
```

```
899   4                 y1 = y1 - 2;
900   4                 x2 = x2 - 1;
901   4                 k = k + 1;
902   4               end;

903   3       qs3:    y1 = 121;
904   3               x2 = 196;

905   3               do while y1 > 115;
906   4                 call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
907   4                 call ulcmd(192,y1,x2,y1);
908   4                 call agcmd( pattern((k - 1) and 3),0ffffh,0ffffh,0ffffh );
909   4                 call ulcmd(192,y1,x2,y1);
910   4                 y1 = y1 - 2;
911   4                 x2 = x2 + 1;
912   4                 k = k + 1;
913   4               end;

914   3       rs3:    y1 = 115;

915   3               do while y1 > 111;
916   4                 call agcmd( pattern(k and 3),0ffffh,0ffffh,0ffffh );
917   4                 call ulcmd(192,y1,223,y1);

918   4                 call agcmd( pattern((k - 1) and 3),0ffffh,0ffffh,0ffffh )
919   4                 call ulcmd(192,y1,223,y1);
920   4                 y1 = y1 - 2;
921   4                 k = k + 1;
922   4               end;
923   3             end;

924   2         end reactor;

                  /*****************************************************************

                                                      MAIN

                  *****************************************************************/

925   1         call init(4096,288,227,2,3,2,3,16,16,12h,82h,80h);
926   1         call reactor;
927   1         call dqexit(0);

928   1       end reactor$module;


MODULE INFORMATION:

     CODE AREA SIZE     = 2859H  10329D
     CONSTANT AREA SIZE = 0027H     39D
     VARIABLE AREA SIZE = 0000H      0D
     MAXIMUM STACK SIZE = 002EH     46D
     1268 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

DICTIONARY SUMMARY:

     360KB MEMORY AVAILABLE
     11KB MEMORY USED   (3%)
     0KB DISK SPACE USED
```